
Ayudantía 10

— Heaps y MST —

Heaps

¿Qué es una cola de prioridad?

- Una estructura de datos que nos permite almacenar datos según su prioridad (dada por un valor). Por ejemplo, pruebas para las que hay que estudiar o urgencia de atención de una persona en una sala de espera
- Un criterio de prioridad: Orden de llegada (FIFO, LIFO). Importa la posición en la cola
- Podría interesarnos otro criterio, pero siempre nos interesará mantener estas funciones:
 - Insertar un dato con prioridad dada
 - Extraer el dato con mayor prioridad
 - Idealmente, cambiar la prioridad de un dato

Array vs Lista ligada

¿Donde almaceno mi cola de prioridad?

Array vs Lista ligada

¿Donde almaceno mi cola de prioridad?

Lo haremos en un array

Array vs Lista ligada

¿Donde almaceno mi cola de prioridad?

Lo haremos en un array

¿Por qué? El tamaño es limitado. Mejor usar una lista

Array vs Lista ligada

¿Donde almaceno mi cola de prioridad?

Lo haremos en un array

¿Por qué? El tamaño es limitado. Mejor usar una lista ligada

Hay casos donde usar una lista ligada nos dará problemas

- Buscar el máximo valor $\rightarrow O(n)$
 - Insertar en la posición correcta/mantener orden $\rightarrow O(n)$
- } caro

Usar arrays nos limitará la cantidad de elementos, pero nos dará facilidad para acceder a los elementos (en $O(1)$) y mantener cierto "orden"

Array vs Lista ligada

¿Donde almaceno mi cola de prioridad?

Lo haremos en un array

¿Por qué? El tamaño es limitado. Mejor usar una lista ligada

Hay casos donde usar una lista ligada nos dará problemas

- Buscar el máximo valor $\rightarrow O(n)$
 - Insertar en la posición correcta/mantener orden $\rightarrow O(n)$
- } caro

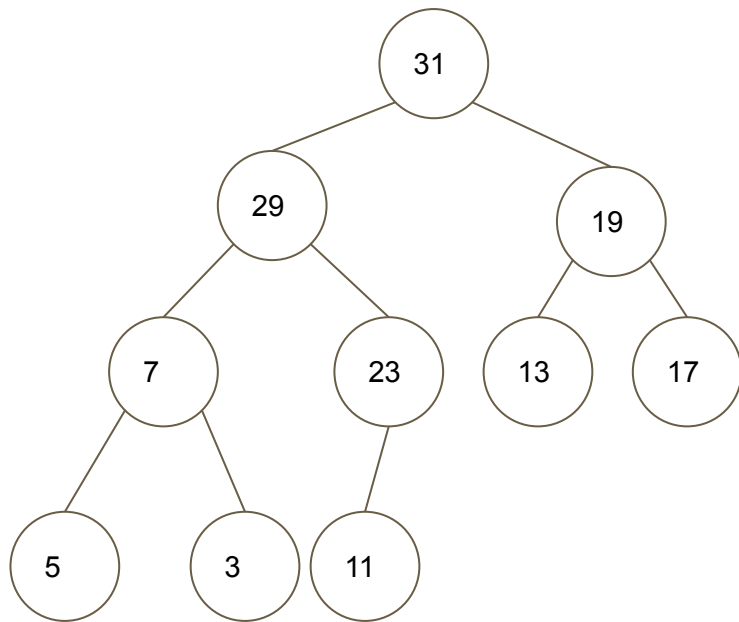
Usar arrays nos limitará la cantidad de elementos, pero nos dará facilidad para acceder a los elementos (en $O(1)$) y mantener cierto "orden"

Para las colas de prioridad usaremos Heaps

¿Qué es un Heap?

- Un heap es un árbol binario que nos permite mantener un **orden de prioridad** los elementos de un conjunto. La prioridad la podemos determinar según un MIN-Heap o un MAX-heap
- A medida que bajamos de nivel, los nodos hijos tendrán menor prioridad que el padre. Entre hermanos no hay ninguna restricción
- Para nuestro objetivo de extraer e insertar de forma eficiente, no necesitamos un orden estricto. Basta tener un orden entre subsectores

Ejemplo de Heap



valor

31

29

19

7

23

13

17

5

3

11

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

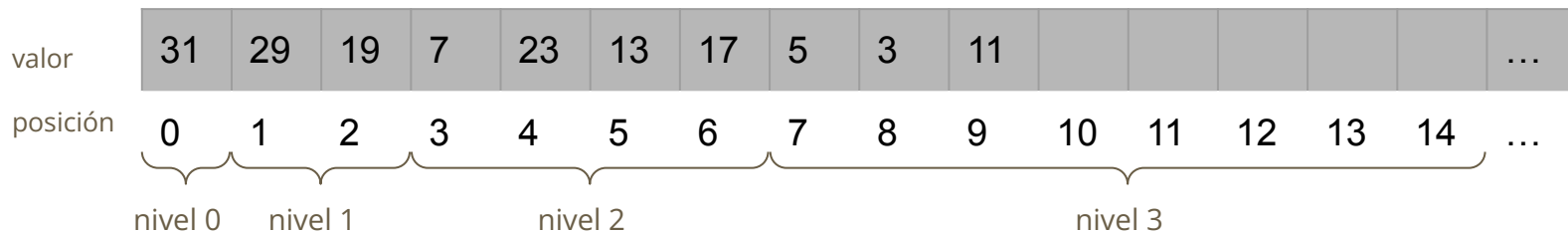
13

14

...

Al ir llenando el árbol por nivel ganamos:

- Minimizar la altura del árbol y compactar en el array
- Una relación entre padres y hijos:
 - El elemento $H[k]$ es padre de $H[2k + 1]$ y $H[2k + 2]$
 - El padre del elemento $H[k]$ es $H[\lfloor (k - 1)/2 \rfloor]$
- agrupar los niveles



A lo que vinimos... extracción e inserción eficiente

1. Efectuamos la operación manteniendo un árbol binario casi-lleno
2. Reestablecemos las propiedades de heap

1) extracción eficiente

Extract(H):

$i \leftarrow$ última celda no vacía de H

$best \leftarrow H[0]$

$H[0] \leftarrow H[i]$

$H[i] \leftarrow \emptyset$

SiftDown($H, 0$)

return $best$

SiftDown(H, i):

if i tiene hijos :

$j \leftarrow$ hijo de i con mayor prioridad

if $H[j] > H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftDown(H, j)

$O(?)$

1) extracción eficiente

Extract(H):

$i \leftarrow$ última celda no vacía de H

$best \leftarrow H[0]$

$H[0] \leftarrow H[i]$

$H[i] \leftarrow \emptyset$

SiftDown($H, 0$)

return $best$

SiftDown(H, i):

if i tiene hijos :

$j \leftarrow$ hijo de i con mayor prioridad

if $H[j] > H[i]$:

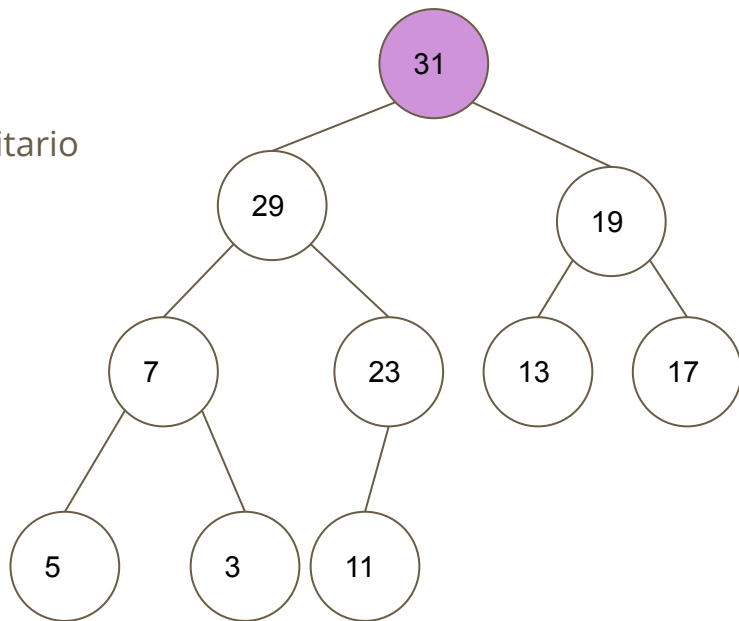
$H[j] \rightleftharpoons H[i]$

SiftDown(H, j)

$O(\log(n))$

1) extracción eficiente

Queremos sacar el
elemento más prioritario



Extract(H):

$i \leftarrow$ última celda no vacía de H

$best \leftarrow H[0]$

$H[0] \leftarrow H[i]$

$H[i] \leftarrow \emptyset$

SiftDown($H, 0$)

return $best$

valor

31

29

19

7

23

13

17

5

3

11

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

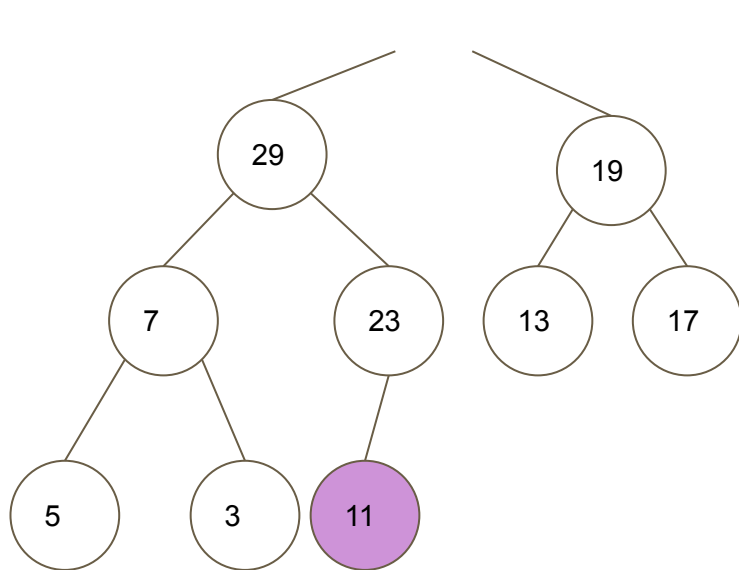
13

14

...

1) extracción eficiente

Extract(H):
 $i \leftarrow$ última celda no vacía de H
 $best \leftarrow H[0]$
 $H[0] \leftarrow H[i]$
 $H[i] \leftarrow \emptyset$
 SiftDown($H, 0$)
 return $best$



31

Extraemos

valor		29	19	7	23	13	17	5	3	11						...
posición	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...

1) extracción eficiente

SiftDown(H, i):

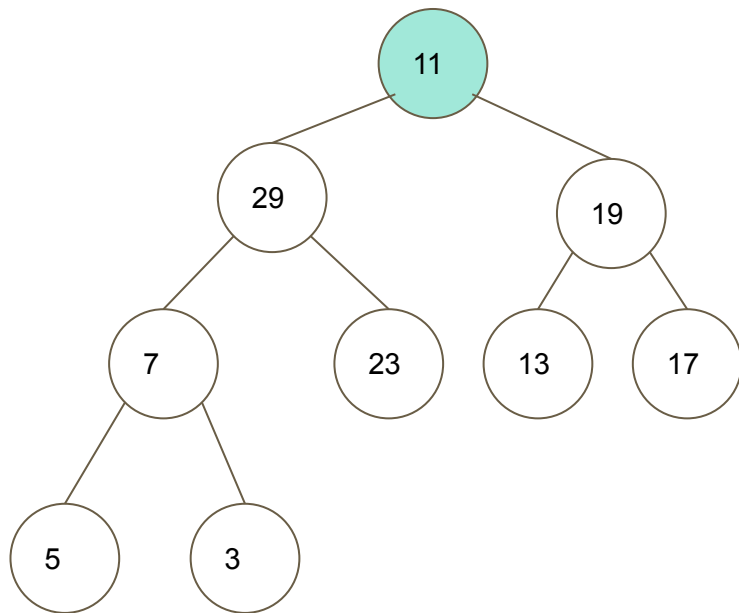
if i tiene hijos :

$j \leftarrow$ hijo de i con mayor prioridad

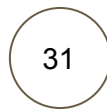
if $H[j] > H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftDown(H, j)



Empezamos con el ShiftDown para reacomodar



valor

11

29

19

7

23

13

17

5

3

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

...

1) extracción eficiente

SiftDown(H, i):

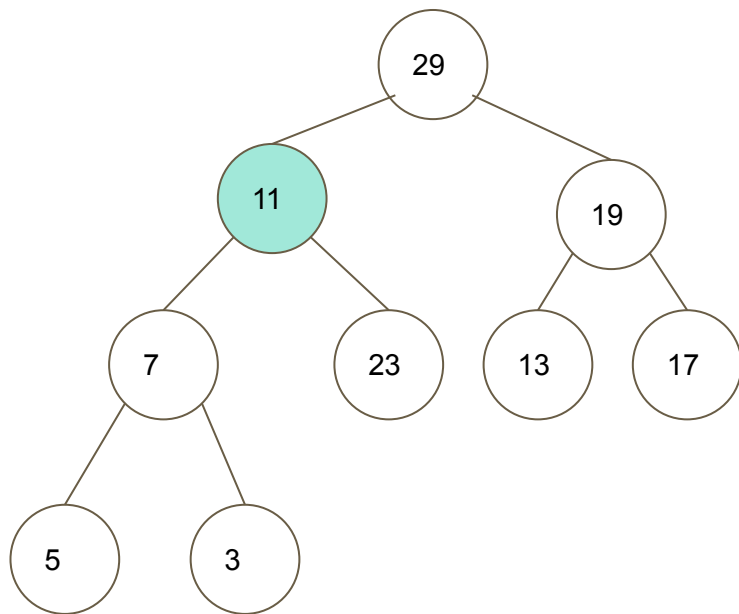
if i tiene hijos :

$j \leftarrow$ hijo de i con mayor prioridad

if $H[j] > H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftDown(H, j)



valor

29

11

19

7

23

13

17

5

3

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

...

1) extracción eficiente

SiftDown(H, i):

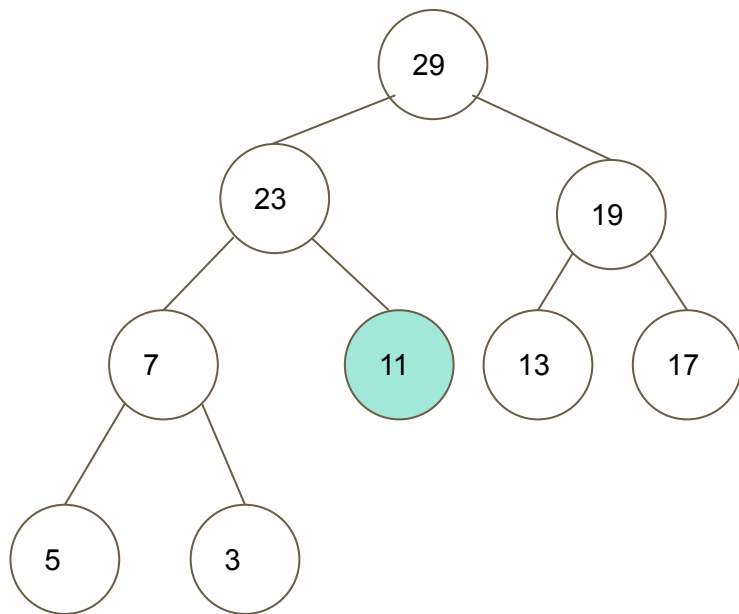
if i tiene hijos :

$j \leftarrow$ hijo de i con mayor prioridad

if $H[j] > H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftDown(H, j)



valor

29

23

19

7

11

13

17

5

3

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

...

2) inserción eficiente

Insert(H):

$i \leftarrow$ primera celda vacía de H

$H[i] \leftarrow e$

SiftUp(H, i)

SiftUp(H, i):

if i tiene padre :

$j \leftarrow \lfloor i/2 \rfloor$

if $H[j] < H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftUp(H, j)

$O(?)$

2) inserción eficiente

Insert(H):

$i \leftarrow$ primera celda vacía de H

$H[i] \leftarrow e$

SiftUp(H, i)

SiftUp(H, i):

if i tiene padre :

$j \leftarrow \lfloor i/2 \rfloor$

if $H[j] < H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftUp(H, j)

$O(\log(n))$

2) inserción eficiente

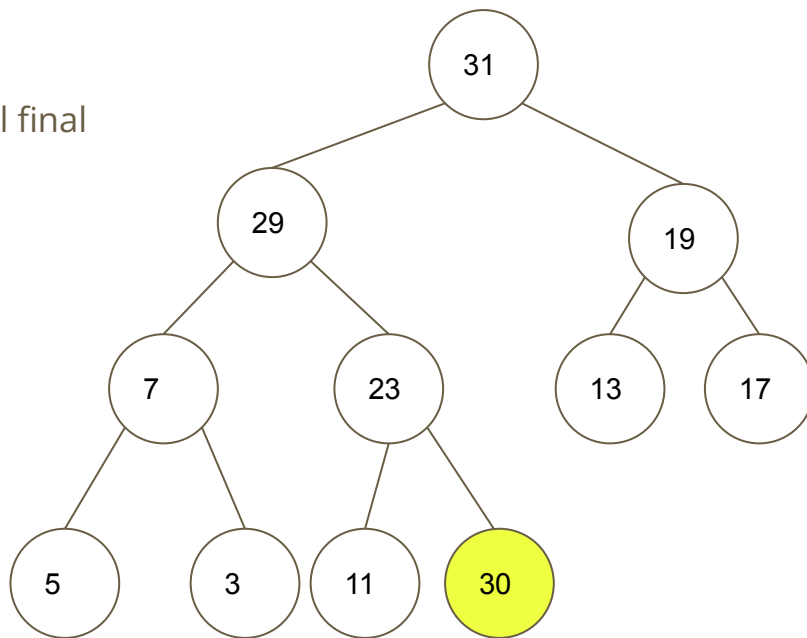
Insertamos, inicialmente, al final
del array al nuevo nodo

Insert(H):

$i \leftarrow$ primera celda vacía de H

$H[i] \leftarrow e$

SiftUp(H, i)



valor

31

29

19

7

23

13

17

5

3

11

30

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

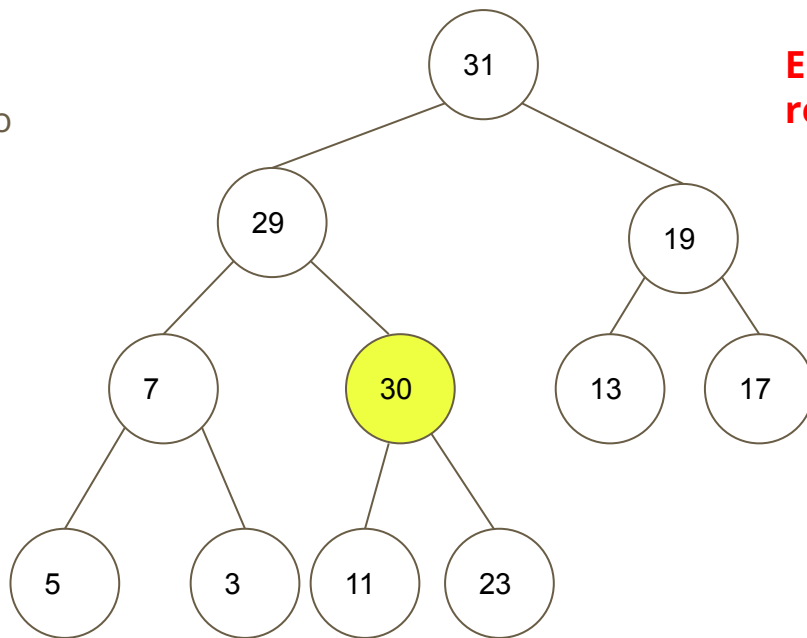
...

2) inserción eficiente

Empezamos a subir el nodo

Empezamos con el ShiftUp para reacomodar

```
SiftUp(H, i):  
  if i tiene padre :  
     $j \leftarrow \lfloor i/2 \rfloor$   
    if  $H[j] < H[i]$  :  
       $H[j] \rightleftharpoons H[i]$   
      SiftUp(H, j)
```



valor

31

29

19

7

30

13

17

5

3

11

23

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

...

2) inserción eficiente

Empezamos a subir el nodo

SiftUp(H, i):

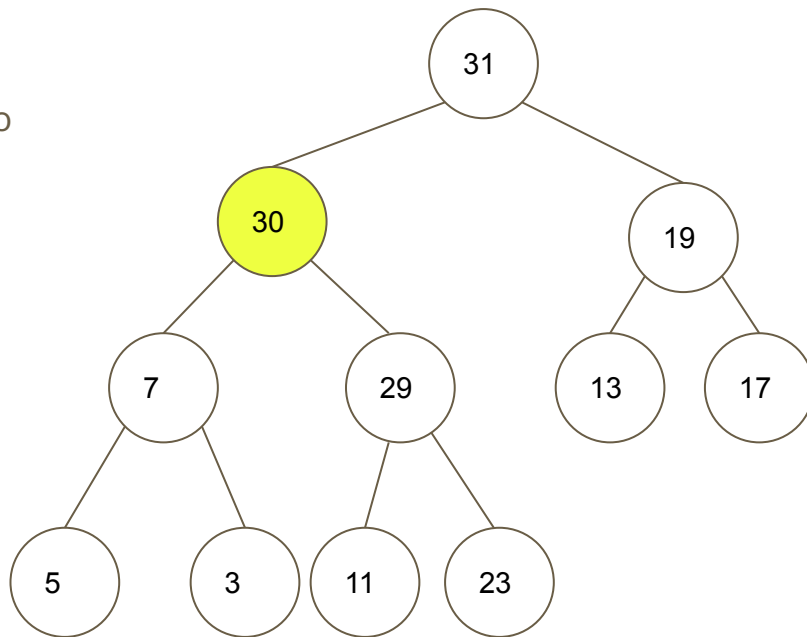
if i tiene padre :

$j \leftarrow \lfloor i/2 \rfloor$

if $H[j] < H[i]$:

$H[j] \rightleftharpoons H[i]$

SiftUp(H, j)



valor

31

30

19

7

29

13

17

5

3

11

23

...

posición

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

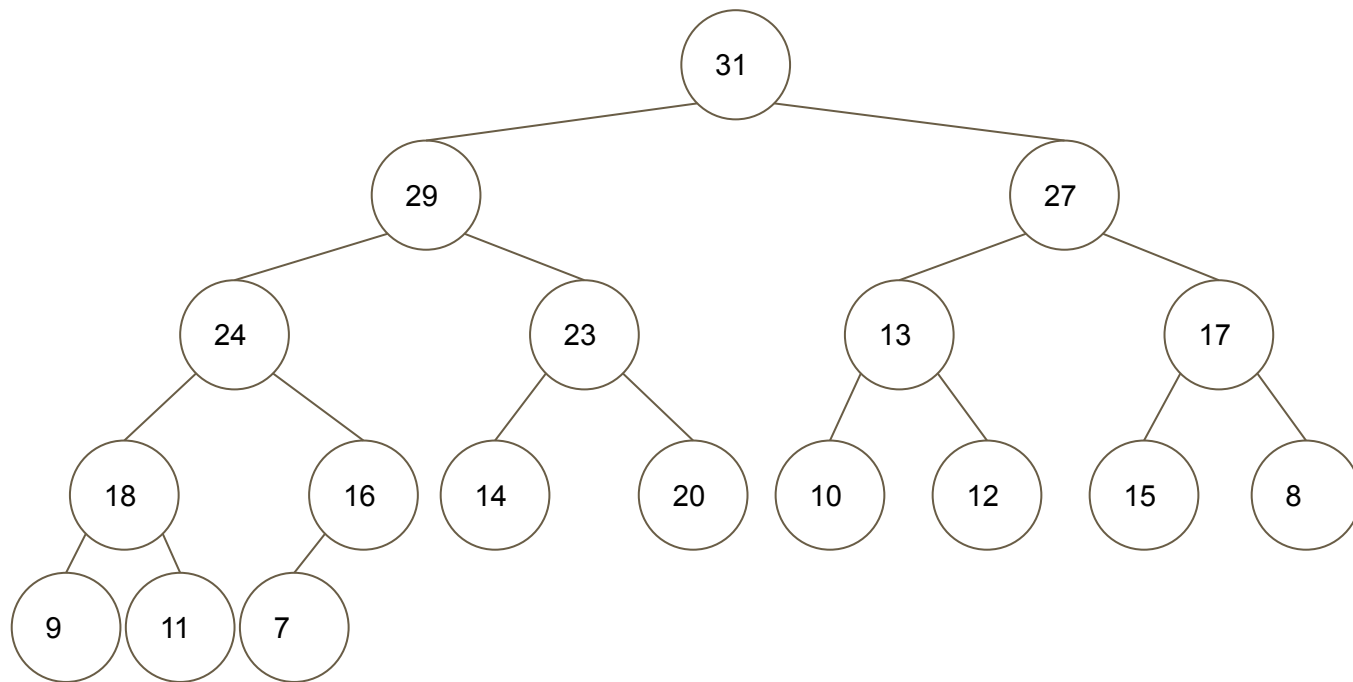
...

¿Qué ganamos?

	Con una lista ligada	Con un array
Extracción del elemento más prioritario	$O(n)$	$O(1)$
Insertar manteniendo orden	$O(n)$	$O(\log(n))$

**¿Y la actualización de
valores?**

Escribe un algoritmo que ordene el Heap en caso de que se actualice el valor del nodo i



valor

31	29	27	24	23	13	17	18	16	14	20	10	12	15	8	9	11	7		
----	----	----	----	----	----	----	----	----	----	----	----	----	----	---	---	----	---	--	--

posición

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----

¿Qué pasa si actualizamos el valor?

Tres casos:

1. El nodo se actualiza con un valor que no supera al padre ni queda por debajo de alguno de los hijos
→ No pasa nada
2. El nodo se actualiza con un valor que supera al del padre
→ Hay que subir el nodo
3. El nodo se actualiza con un valor que lo hace quedar por debajo de alguno de los hijos
→ Hay que bajar el nodo y cambiarlo por ese hijo

input: Heap, índice del nodo a actualizar, nuevo valor del nodo

Update(H, i, v):

H[i] = v

if H tiene padre **and** $H[i] > H[\lfloor (i-1)/2 \rfloor]$:

ShiftUp(H, i)

else if H tiene hijos **and** H tiene menor prioridad que alguno de ellos:

ShiftDown(H, i)

I3 2023-1 P1

La atención al afiliado en la sucursal de una Isapre requiere que, para ser atendidos, éstos deben registrar su RUT y el tipo de atención solicitada (venta de bonos, reembolsos, pago de cotizaciones, cambio de planes, etc.) en un tótem de auto-atención . Este entrega un número secuencial ascendente, los que son invocados en orden por los ejecutivos de la sucursal al estar disponibles para atender al afiliado.

I3 2023-1 P1

La atención al afiliado en la sucursal de una Isapre requiere que, para ser atendidos, éstos deben registrar su RUT y el tipo de atención solicitada (venta de bonos, reembolsos, pago de cotizaciones, cambio de planes, etc.) en un tótem de auto-atención . Este entrega un número secuencial ascendente, los que son invocados en orden por los ejecutivos de la sucursal al estar disponibles para atender al afiliado.

(a) El tótem utiliza la función `new(A, b, tipo, sec)` para registrar en **A** los datos del afiliado **b** que solicita atención con secuencia `sec` entregada por el tótem . El sistema utiliza la llamada `b = next(A)` para sacar de **A** al siguiente afiliado **b** a atender según la secuencia. Escriba el pseudocódigo de ambas funciones e indique explícitamente qué estructura de datos utiliza para representar **A**.

I3 2023-1 P1

(a) El tótem utiliza la función **new(A, b, tipo, sec)** para registrar en **A** los datos del afiliado **b** que solicita atención con secuencia **sec** entregada por el tótem . El sistema utiliza la llamada **b = next(A)** para sacar de **A** al siguiente afiliado **b** a atender según la secuencia. Escriba el pseudocódigo de ambas funciones e indique explícitamente qué estructura de datos utiliza para representar **A**.

SOLUCIÓN:

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

```
new(A, b, tipo, sec):  
    b.tipo ← tipo  
    b.sec ← sec  
    Insert(A, b)
```

```
next(A) :  
    return Remove(A, 0)
```

donde **Insert(A, b)** agrega **b** al final de la lista ligada **A**, y **Remove(A, 0)** retorna el primer elemento (cabeza) de la lista ligada **A** luego de extraerlo.

I3 2023-1 P1

La atención al afiliado en la sucursal de una Isapre requiere que, para ser atendidos, éstos deben registrar su RUT y el tipo de atención solicitada (venta de bonos, reembolsos, pago de cotizaciones, cambio de planes, etc.) en un tótem de auto-atención . Este entrega un número secuencial ascendente, los que son invocados en orden por los ejecutivos de la sucursal al estar disponibles para atender al afiliado.

(b) La gerente de la oficina desea mejorar la atención de los adultos mayores (> 65 años), considerando que al registrar al afiliado es posible incluir en **b** su edad (en meses según **b.edad**). Pide entonces promover que los adultos mayores sean atendidos primero, de mayor a menor edad, y luego los demás afiliados según la secuencia entregada por el tótem. Modifique el pseudocódigo de las funciones de (a) e indique explícitamente qué estructura de datos usa para **A**. Puede suponer que no hay dos afiliados con la misma edad en meses.

Pista: La secuencia del tótem es creciente no negativa

I3 2023-1 P1

(b) La gerente de la oficina desea mejorar la atención de los adultos mayores (> 65 años), considerando que al registrar al afiliado es posible incluir en **b** su edad (en meses según **b.edad**). Pide entonces promover que los adultos mayores sean atendidos primero, de mayor a menor edad, y luego los demás afiliados según la secuencia entregada por el tótem. Modifique el pseudocódigo de las funciones de (a) e indique explícitamente qué estructura de datos usa para **A**. Puede suponer que no hay dos afiliados con la misma edad en meses.

Pista: La secuencia del tótem es creciente no negativa

SOLUCIÓN:

En este caso, se utiliza un MIN-heap donde la prioridad se almacena en el atributo **sec**. Para adultos mayores, este valor se reemplazará por su edad en negativo, de forma que el atributo **sec** por sí solo sirve como prioridad del MIN-heap. Con esto, los métodos quedan...

I3 2023-1 P1

SOLUCIÓN:

En este caso, se utiliza un MIN-heap donde la prioridad se almacena en el atributo **sec**. Para adultos mayores, este valor se reemplazará por su edad en negativo, de forma que el atributo **sec** por sí solo sirve como prioridad del MIN-heap. Con esto, los métodos quedan...

```
new(A, b, tipo, sec):  
    b.tipo ← tipo  
    if b.edad < 65 * 12:  
        b.sec ← sec  
    else:  
        b.sec ← -1 * b.edad  
    Insert(A, b)
```

```
next(A) :  
    return Extract(A, 0)
```

donde **Extract(A)** es el método de extracción de raíz en MIN-heap.

I3 2023-1 P1

La atención al afiliado en la sucursal de una Isapre requiere que, para ser atendidos, éstos deben registrar su RUT y el tipo de atención solicitada (venta de bonos, reembolsos, pago de cotizaciones, cambio de planes, etc.) en un tótem de auto-atención . Este entrega un número secuencial ascendente, los que son invocados en orden por los ejecutivos de la sucursal al estar disponibles para atender al afiliado.

(c) Al medir el comportamiento de la mejora anterior se observa que el tiempo promedio de espera para ser atendido depende del tipo de atención de quienes llegaron antes, ya que algunas atenciones son rápidas (venta de bonos) y otras lentas (cambio de planes). Un consultor propone aplicar la estrategia SJF (Short Job First, Tarea Más Corta Primero) para privilegiar a los que requieren atenciones breves. Así, se puede usar un factor $0 \leq f \leq 1$ donde 1 representa la tarea más breve. Indique (no se requiere el pseudocódigo) qué debería modificar en su respuesta de (b) para incorporar este cambio.

I3 2023-1 P1

(c) Al medir el comportamiento de la mejora anterior se observa que el tiempo promedio de espera para ser atendido depende del tipo de atención de quienes llegaron antes, ya que algunas atenciones son rápidas (venta de bonos) y otras lentas (cambio de planes). Un consultor propone aplicar la estrategia SJF (Short Job First, Tarea Más Corta Primero) para privilegiar a los que requieren atenciones breves. Así, se puede usar un factor $0 \leq f \leq 1$ donde 1 representa la tarea más breve. Indique (no se requiere el pseudocódigo) qué debería modificar en su respuesta de (b) para incorporar este cambio.

SOLUCIÓN:

Se debe actualizar la forma en que se calcula la prioridad en el método new. Una estrategia es dividir la prioridad por el factor f de manera que las tareas más largas tienen mayores valores de prioridad luego del cambio.

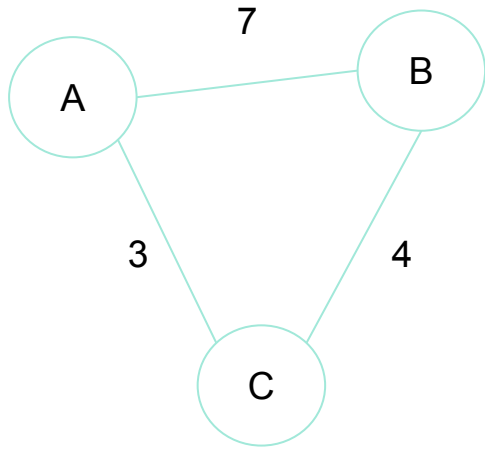
MST

MST

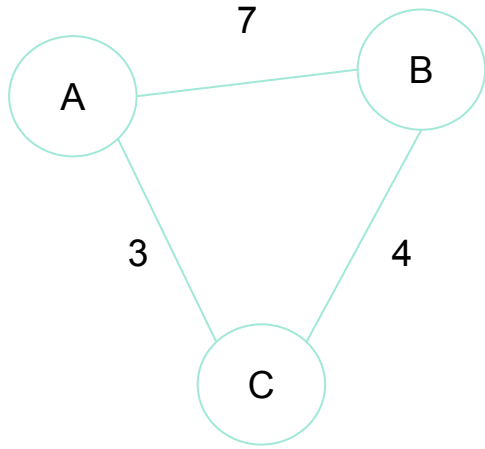
Definición

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

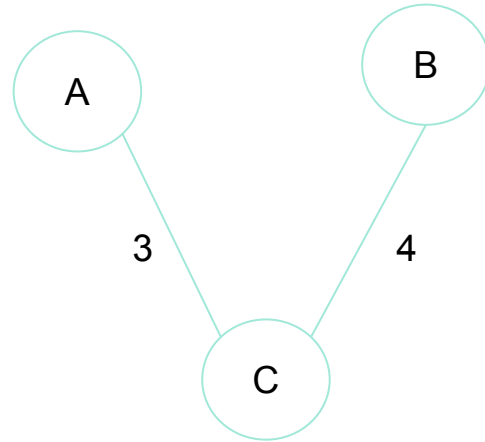
Ejemplos rápidos #1



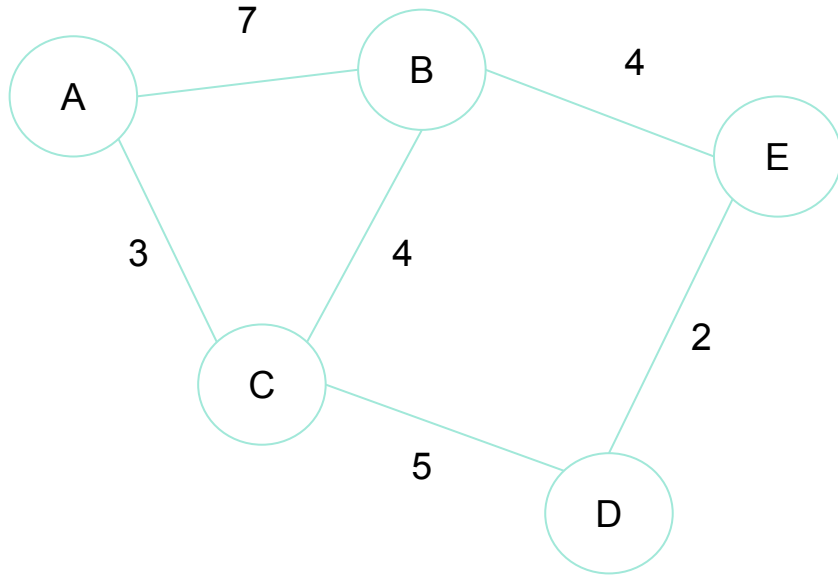
Ejemplos rápidos #1



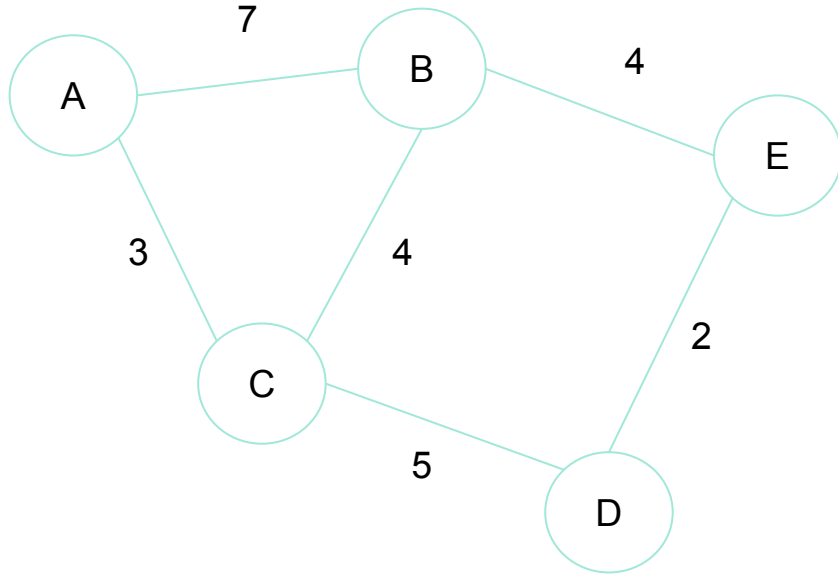
MST
Costo = 7



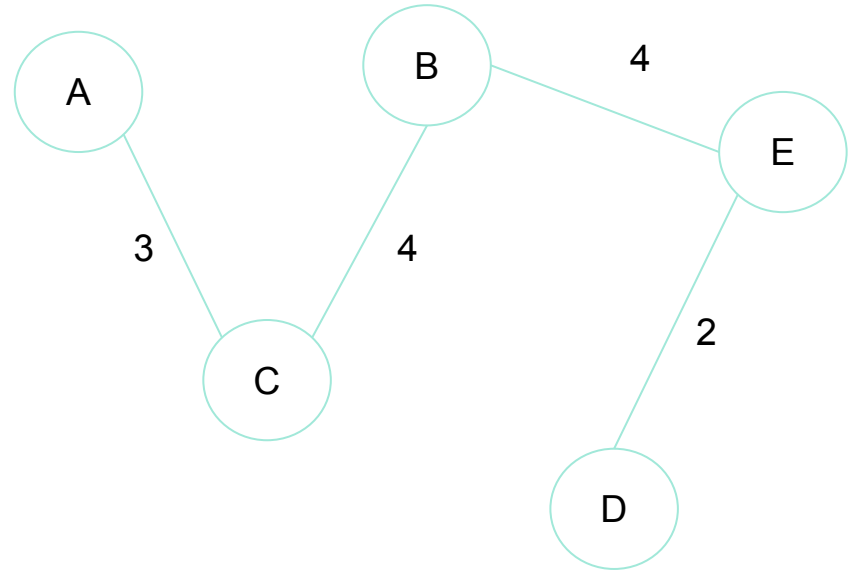
Ejemplos rápidos #2



Ejemplos rápidos #2



MST
Costo = 13



Problema

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

Soluciones

- Algoritmo de Prim
- Algoritmo de Kruskal

Soluciones

- Algoritmo de Prim
- **Algoritmo de Kruskal**

Kruskal

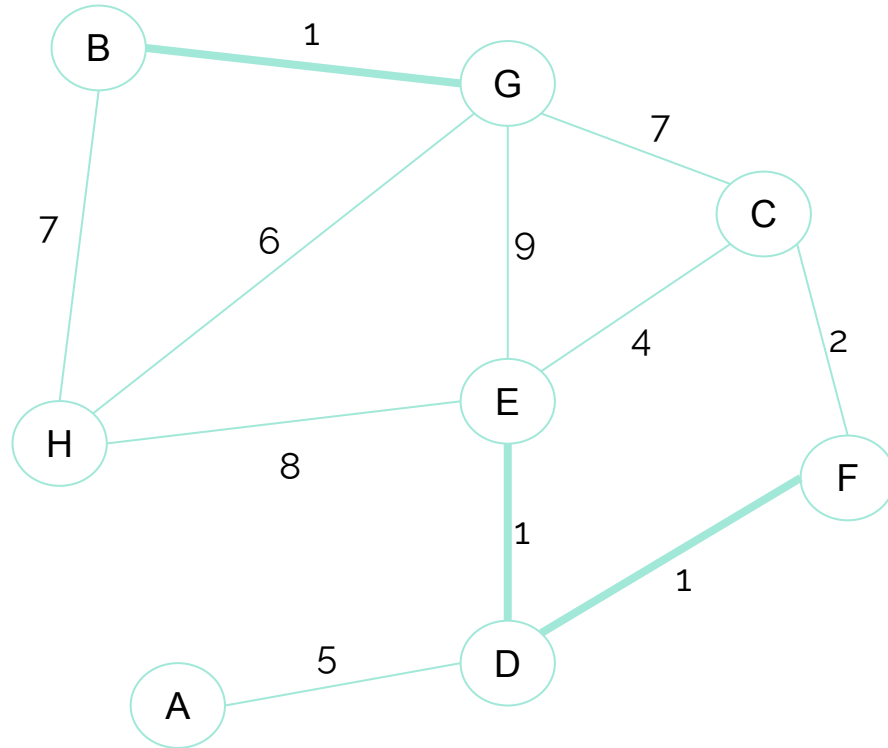
Idea base: Crear un bosque que converge en un único árbol

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?

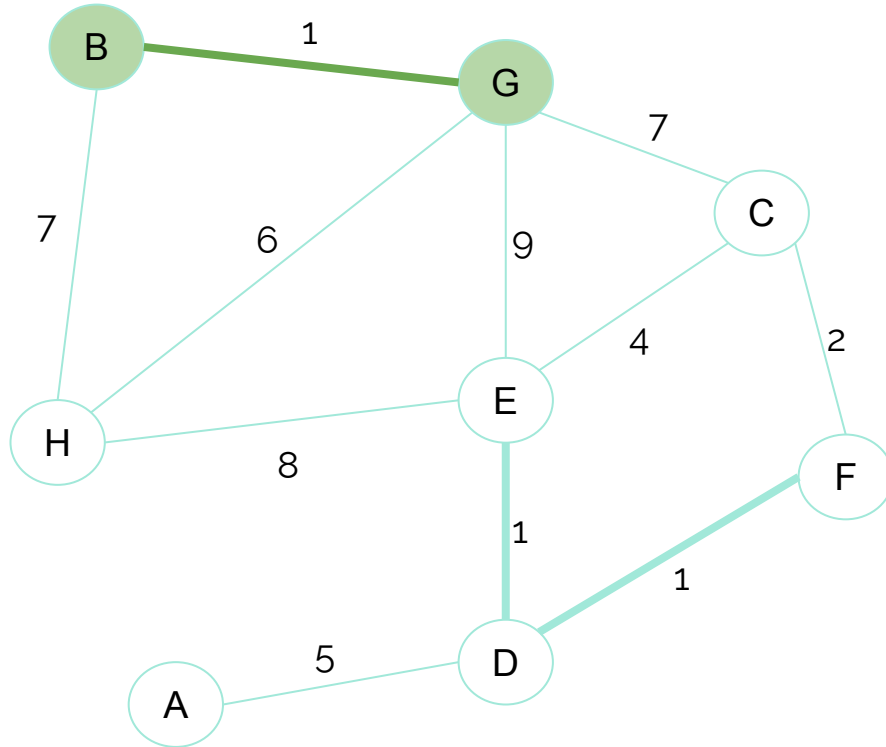
Ejemplo:



— Aristas candidatas

— Aristas en MST

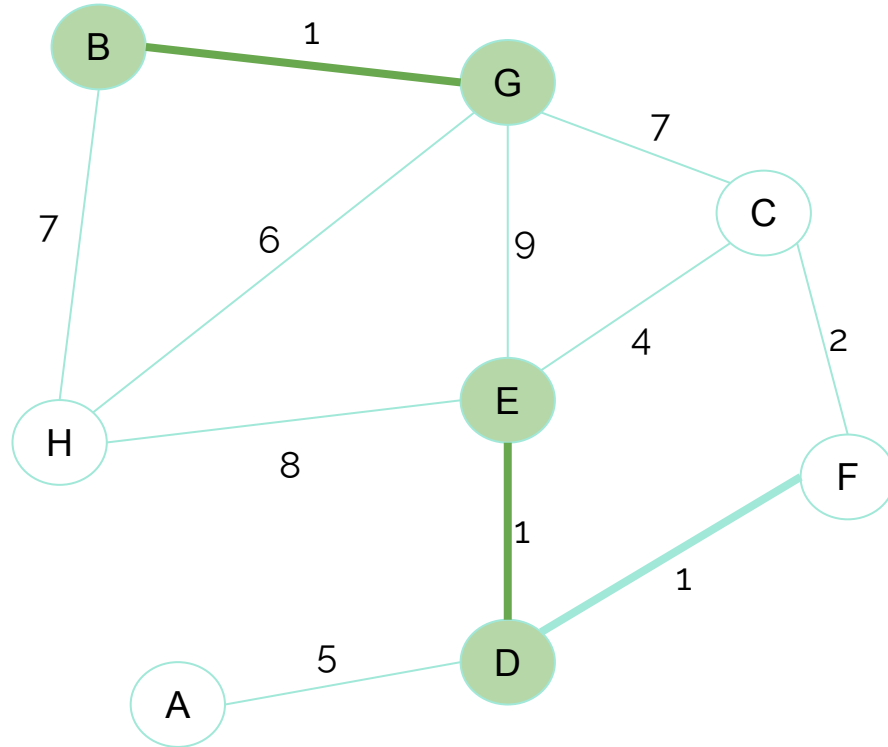
Ejemplo:



— Aristas candidatas

— Aristas en MST

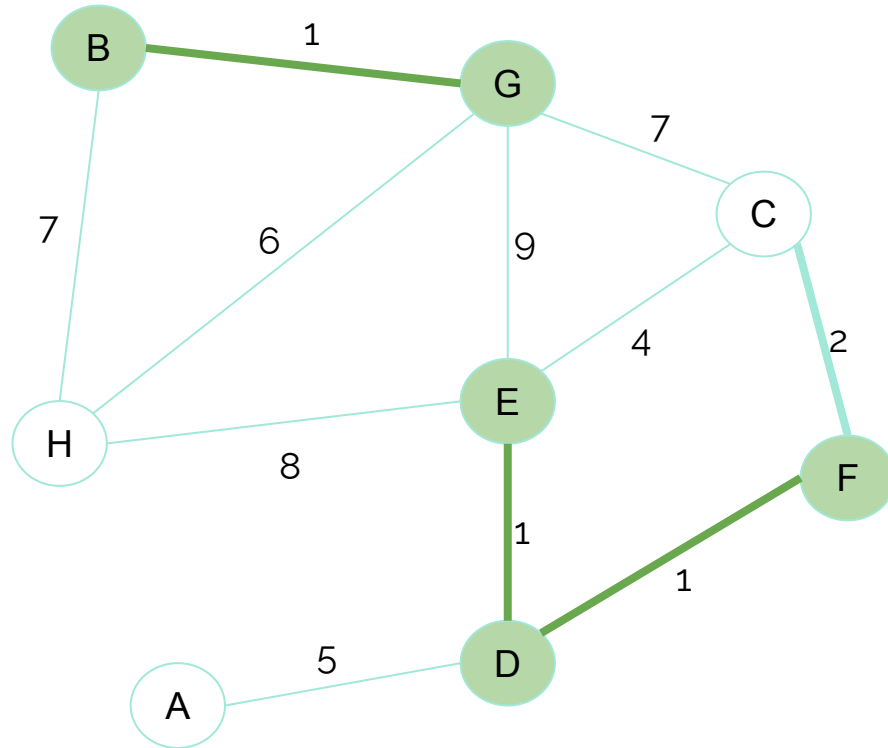
Ejemplo:



— Aristas candidatas

— Aristas en MST

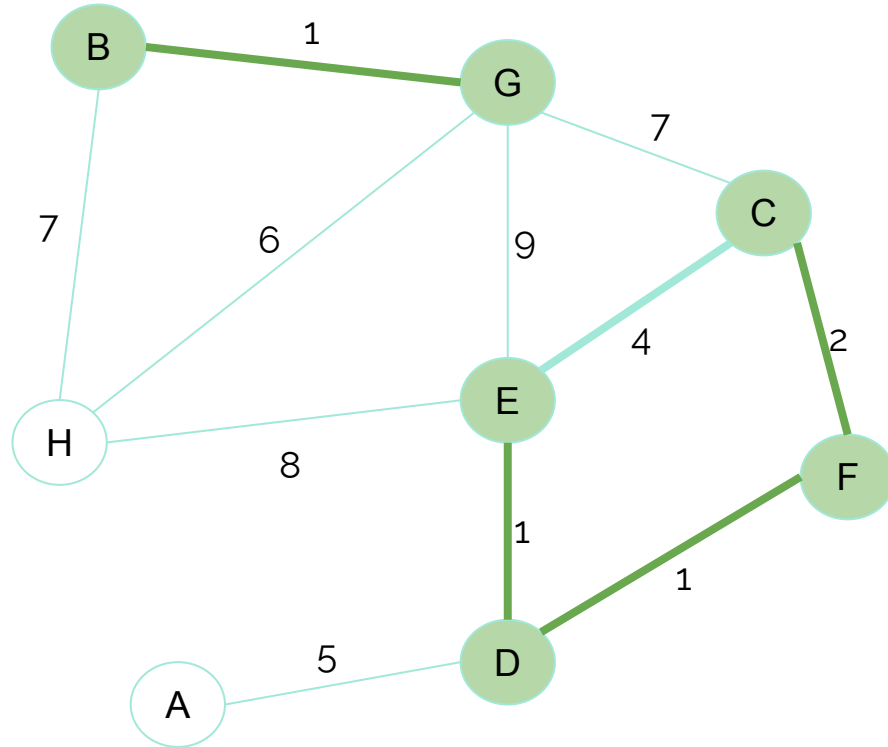
Ejemplo:



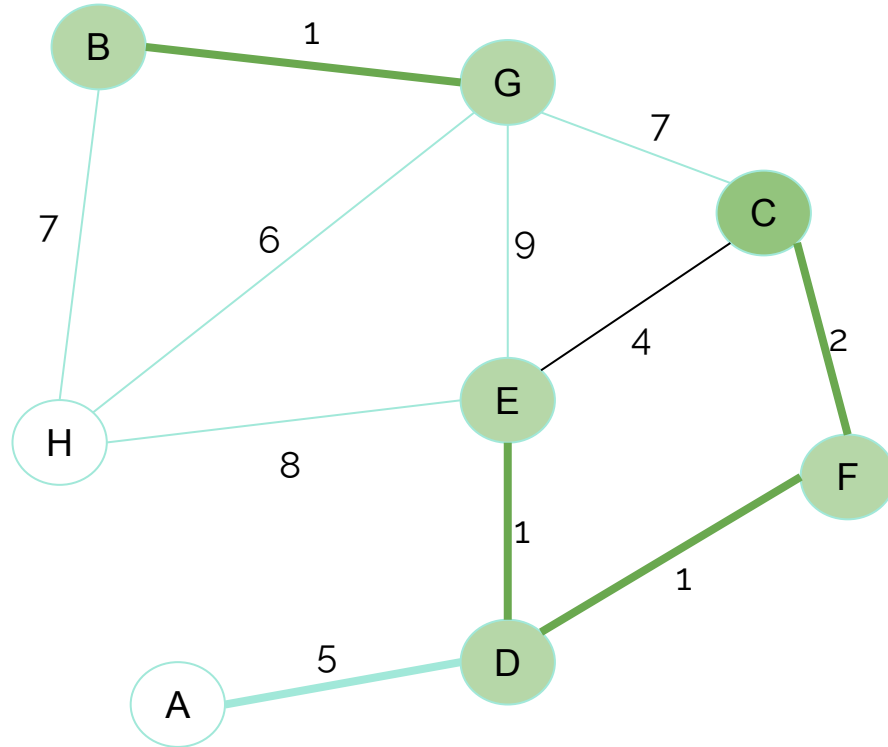
— Aristas candidatas

— Aristas en MST

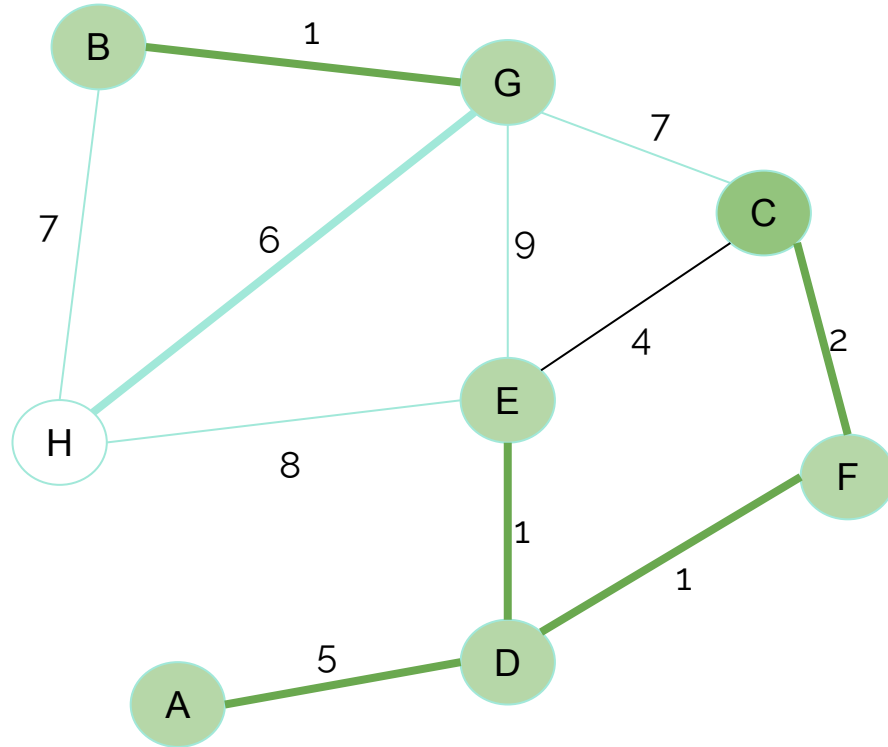
Ejemplo:





Ejemplo:



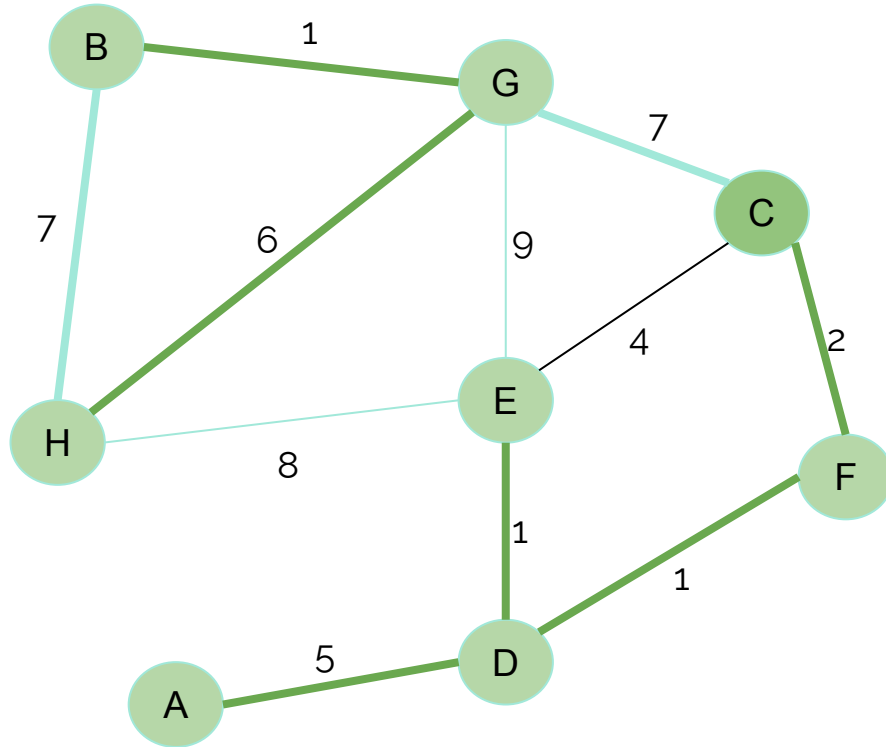
Ejemplo:



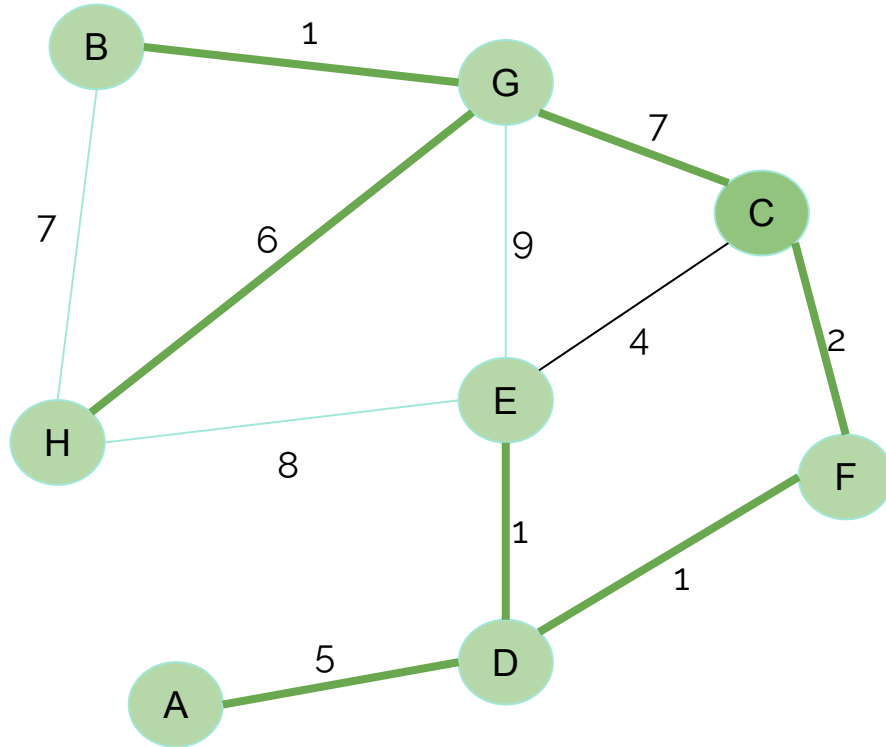
 Aristas candidatas

 Aristas en MST

Ejemplo:



Ejemplo:



— Aristas candidatas

— Aristas en MST

Kruskal

kruskal(Grafo G)

 UnionFind T

 E = G.edges

 sort(E) //Por el peso

 for w,u,v in E

 if not T.same_set(u,v)

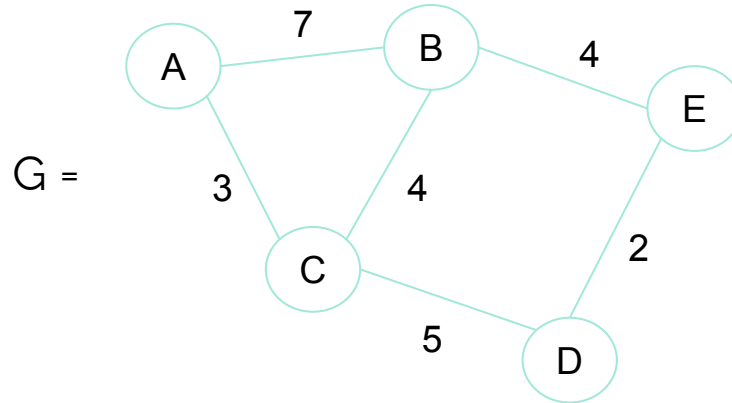
 T.join(u,v)

 //añadir arista u,v

Algoritmo de Prim: conceptos previos

- **Corte:** Partición (V_1, V_2) de $V(G)$
 - V_1 y V_2 no son vacíos
 - La **unión** de V_1 y $V_2 = V(G)$
 - V_1 y V_2 no comparten vértices

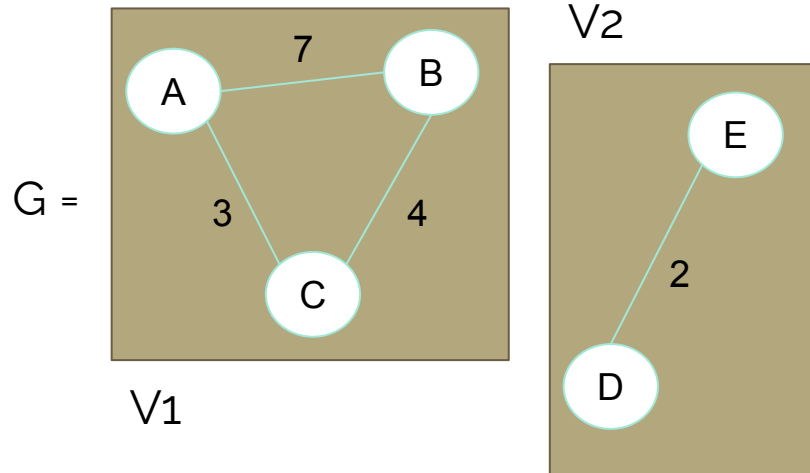
¿Cómo se ve un corte?



Algoritmo de Prim: conceptos previos

- **Corte:** Partición (V_1, V_2) de $V(G)$
 - V_1 y V_2 no son vacíos
 - La **unión** de V_1 y $V_2 = V(G)$
 - V_1 y V_2 no comparten vértices

¿Cómo se ve un corte?

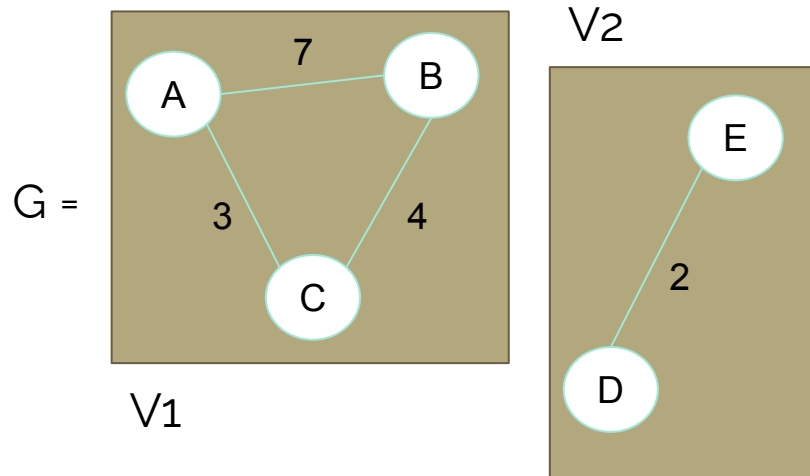


Algoritmo de Prim: conceptos previos

- Arista que **cruza un corte** (V_1, V_2):

Arista que comienza en un vértice de V_1 y termina en un vértice de V_2

¿Cómo se ve?

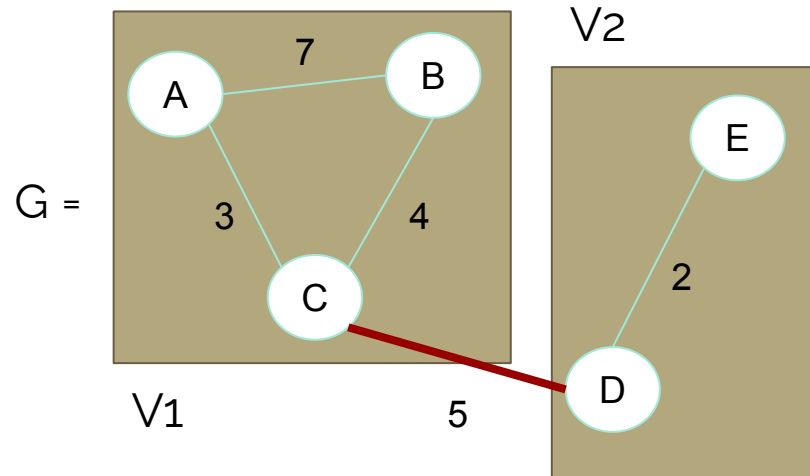


Algoritmo de Prim: conceptos previos

- Arista que **cruza un corte** (V_1, V_2):

Arista que comienza en un vértice de V_1 y termina en un vértice de V_2

¿Cómo se ve?



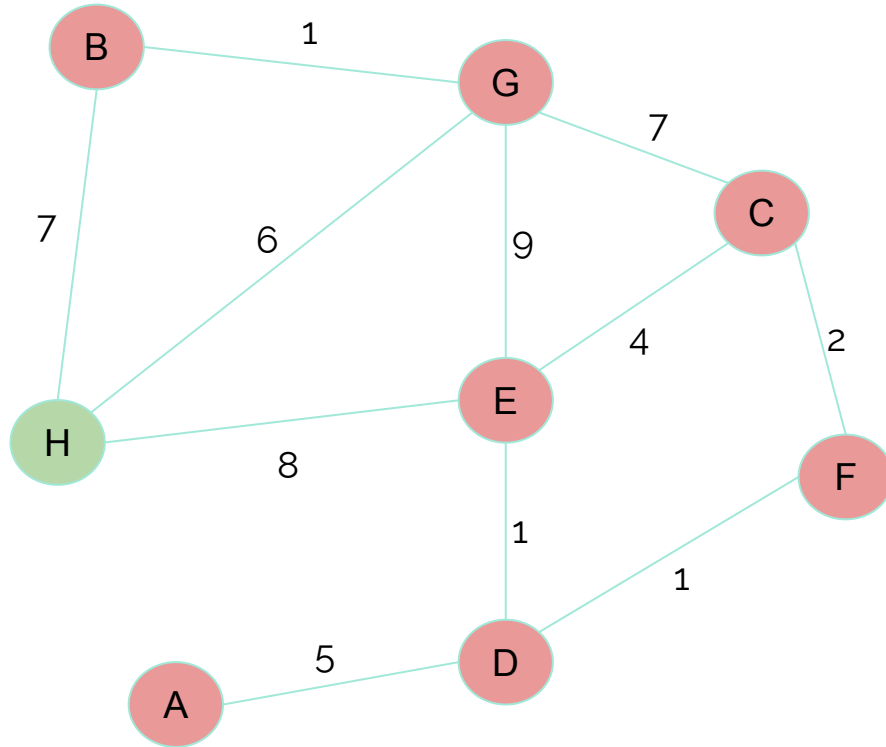
Algoritmo de Prim:

- Idea base: Utilizar aristas que cruzan cortes para guiar su construcción

Sea $G = (V, E)$ grafo no dirigido y v nodo inicial cualquiera.

1. $R = \{v\}$ y $R' = V - R$ # Corte de v y todos los nodos restantes
2. Sea e la arista de menor costo que cruza R a R' # Usamos arista que cruza corte
3. Sea u el nodo de e que pertenece a R'
4. Agregar e al MST. Eliminar u de R' y agregar a R # Ahora tenemos un nuevo corte
5. Si quedan elementos en R' , volver al paso 2

Ejemplo:

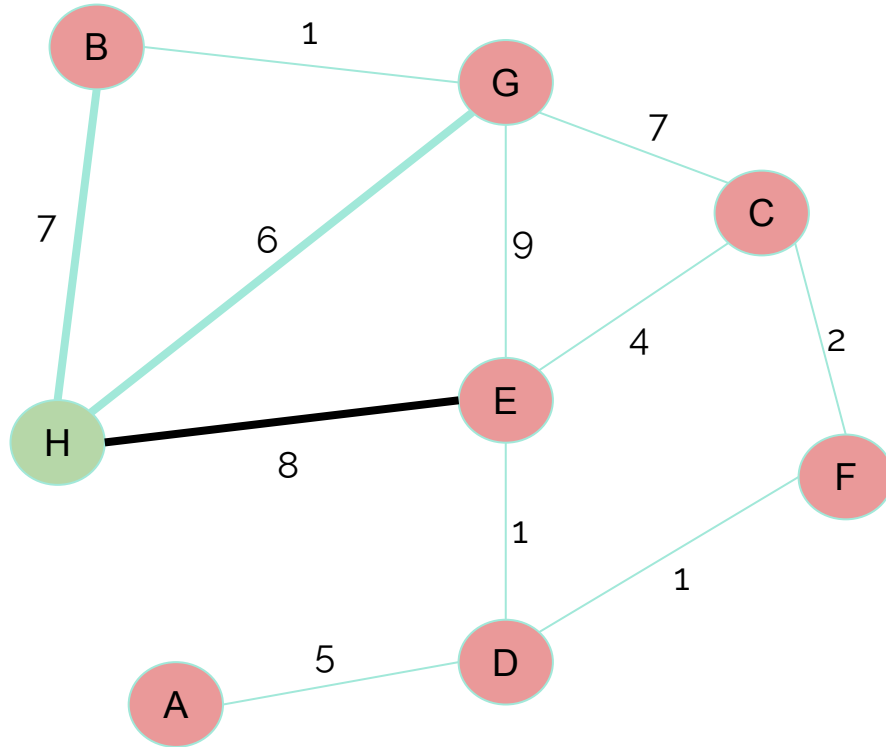


$V = H$

$R = \{ H \}$

$R' = \{ A, B, C, D, E, F, G \}$

Ejemplo:



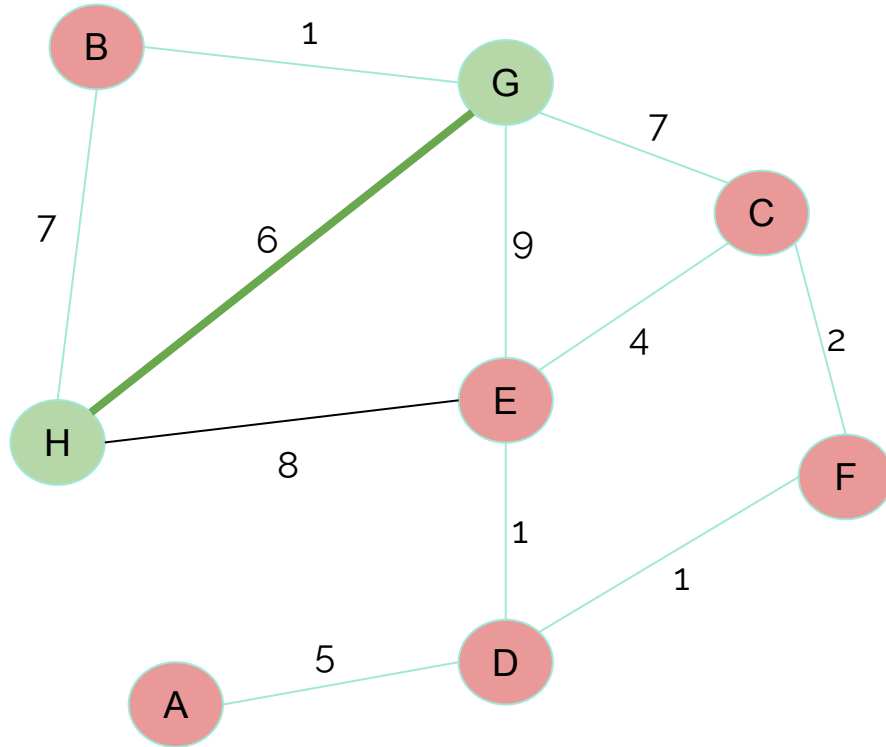
$V = H$

$R = \{ H \}$

$R' = \{ A, B, C, D, E, F, G \}$

 Aristas candidatas

Ejemplo:



$V = H$

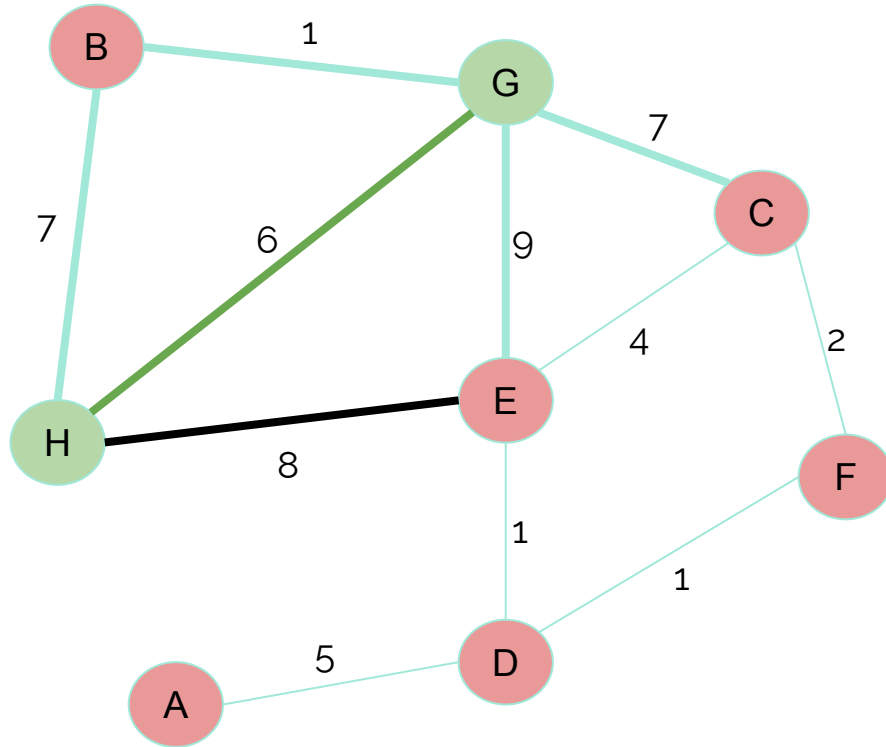
$R = \{ H, G \}$

$R' = \{ A, B, C, D, E, F \}$

— Aristas
candidatas

— Aristas en MST

Ejemplo:



$V = H$

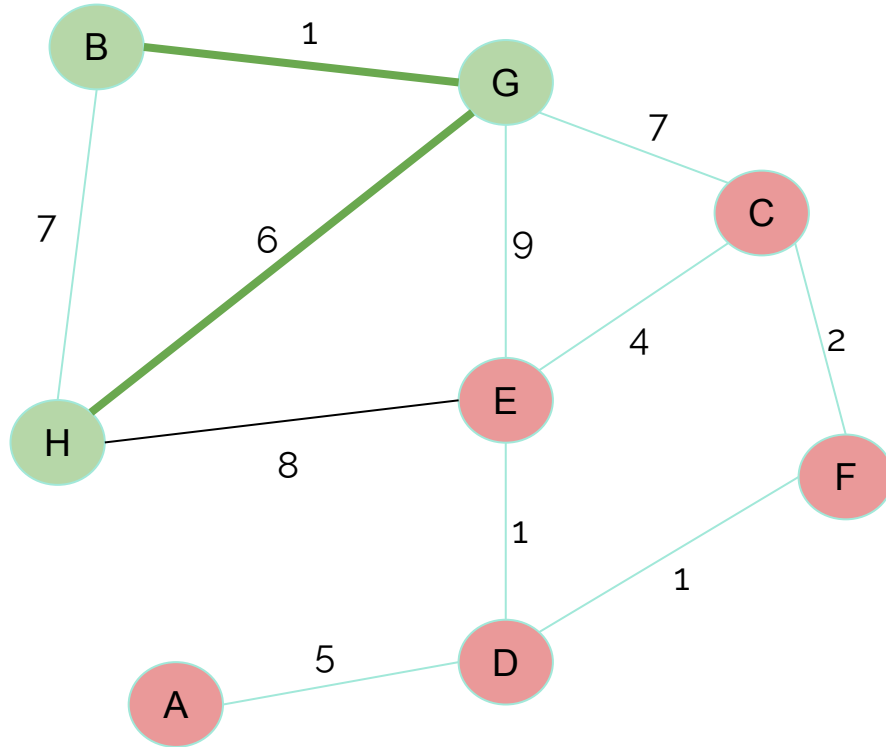
$R = \{ H, G \}$

$R' = \{ A, B, C, D, E, F \}$

— Aristas
candidatas

— Aristas en MST

Ejemplo:



$V = H$

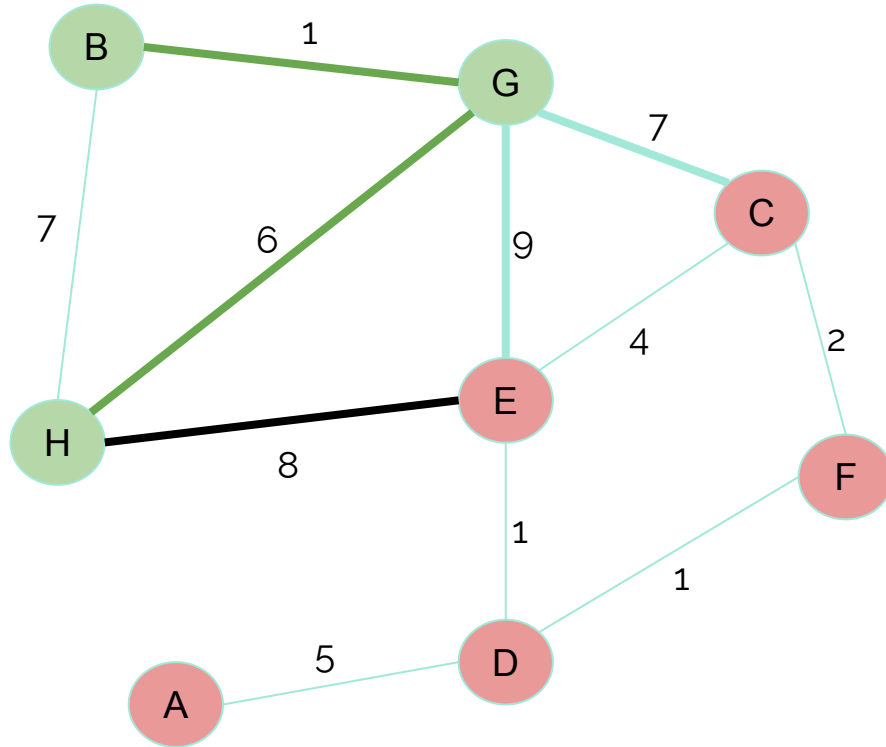
$R = \{ H, G, B \}$

$R' = \{ A, C, D, E, F \}$

 Aristas candidatas

 Aristas en MST

Ejemplo:



$V = H$

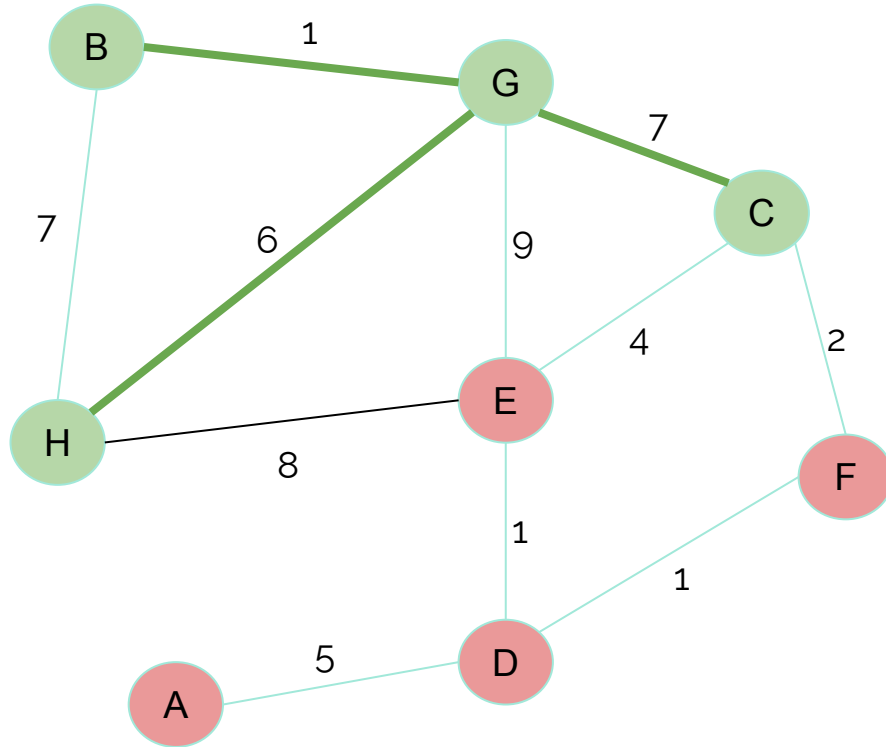
$R = \{ H, G, B \}$

$R' = \{ A, C, D, E, F \}$

— Aristas
candidatas

— Aristas en MST

Ejemplo:



$V = H$

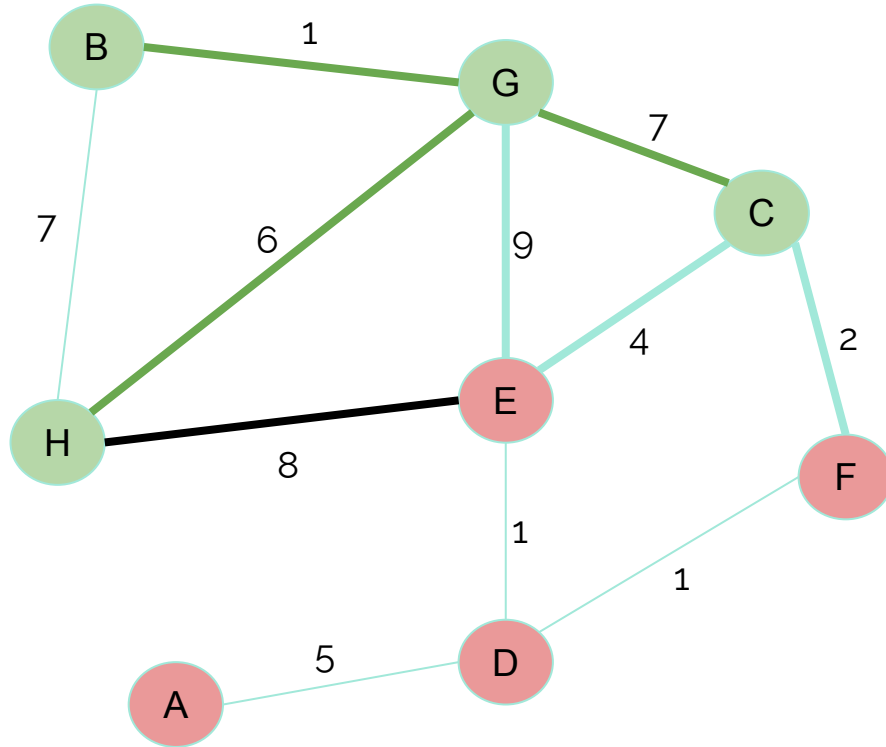
$R = \{ H, G, B, C \}$

$R' = \{ A, D, E, F \}$

 Aristas candidatas

 Aristas en MST

Ejemplo:



$V = H$

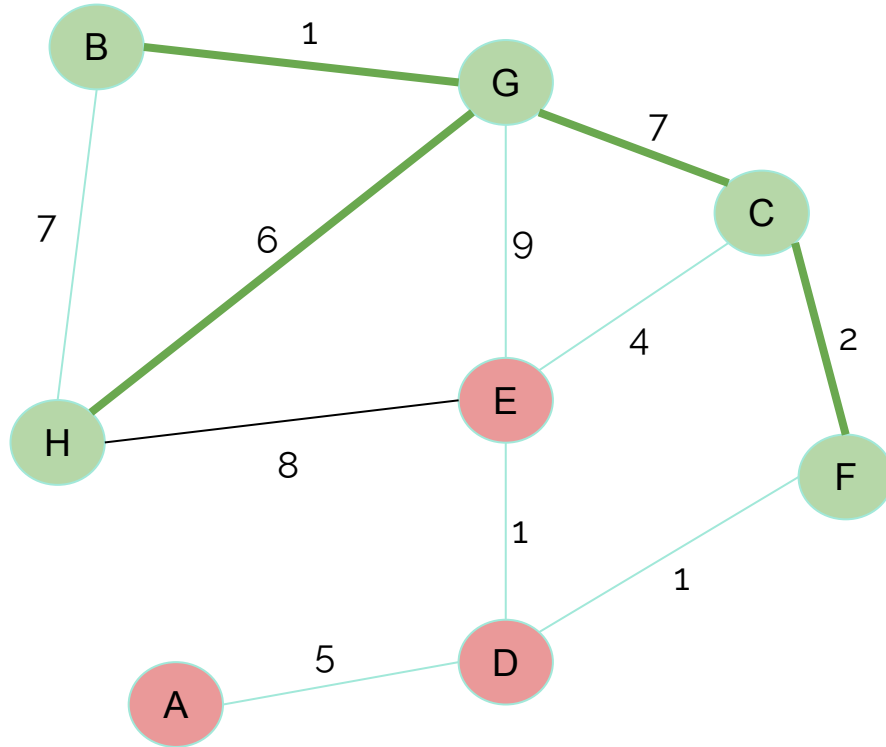
$R = \{ H, G, B, C \}$

$R' = \{ A, D, E, F \}$

— Aristas candidatas

— Aristas en MST

Ejemplo:



$V = H$

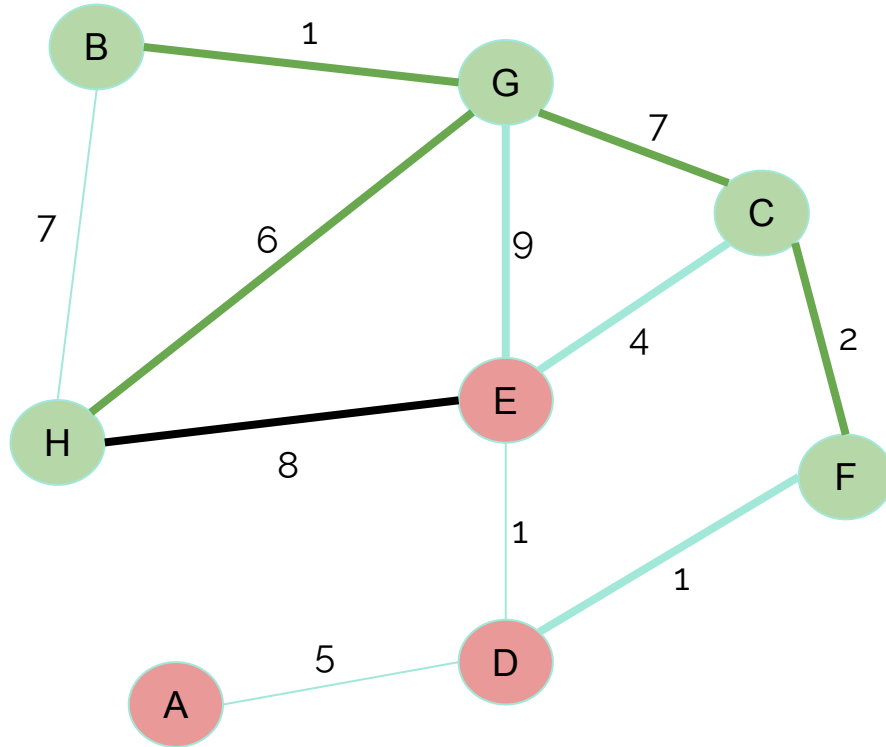
$R = \{H, G, B, C, F\}$

$R' = \{A, D, E\}$

 Aristas candidatas

 Aristas en MST

Ejemplo:



$V = H$

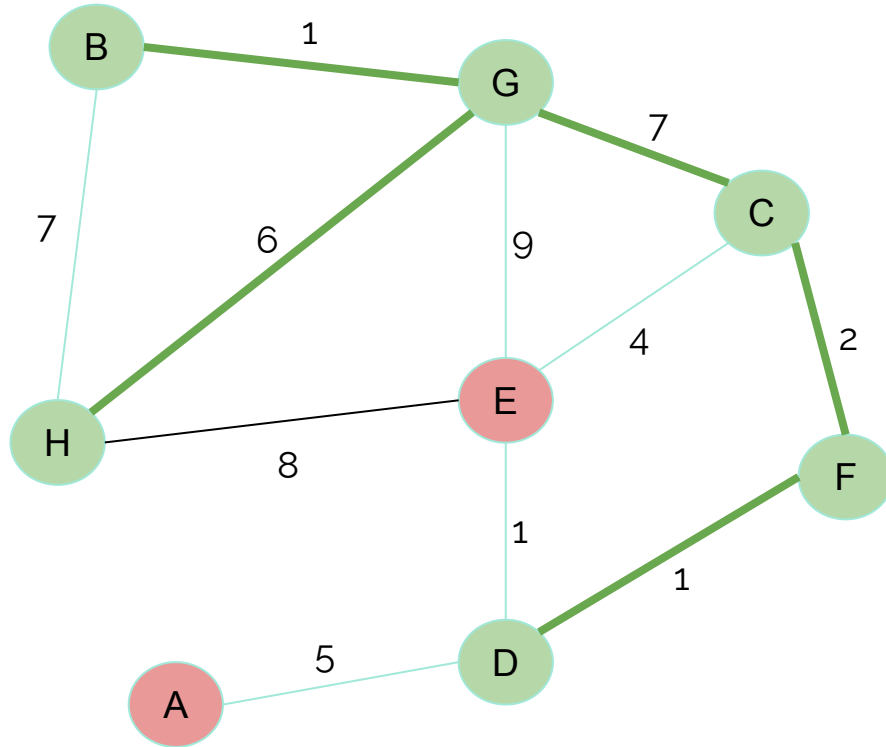
$R = \{H, G, B, C, F\}$

$R' = \{A, D, E\}$

 Aristas candidatas

 Aristas en MST

Ejemplo:



$V = H$

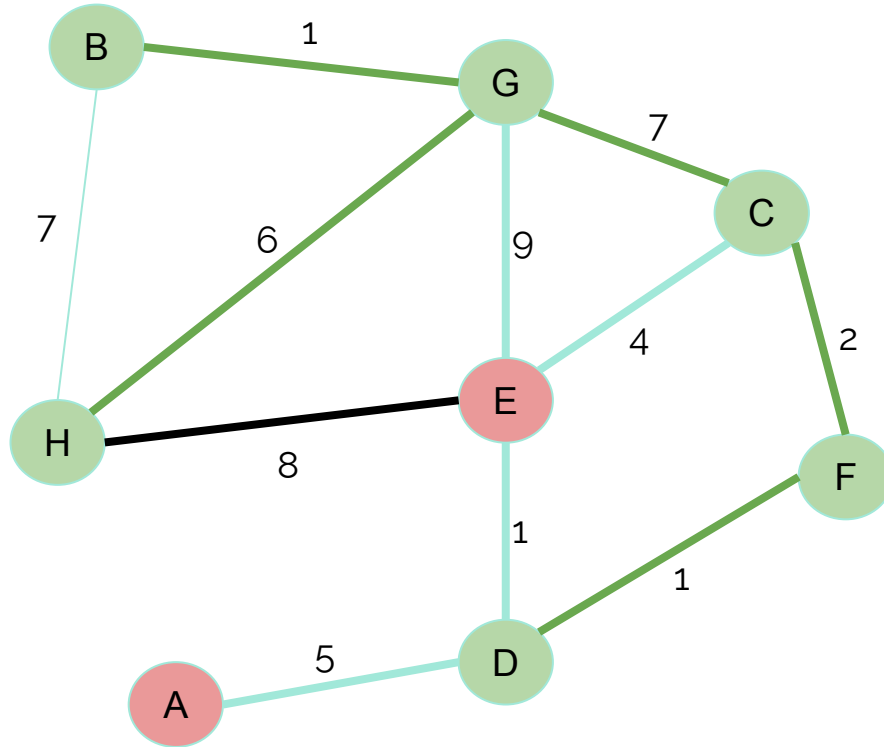
$R = \{ H, G, B, C, F, D \}$

$R' = \{ A, E \}$

— Aristas
candidatas

— Aristas en MST

Ejemplo:



$V = H$

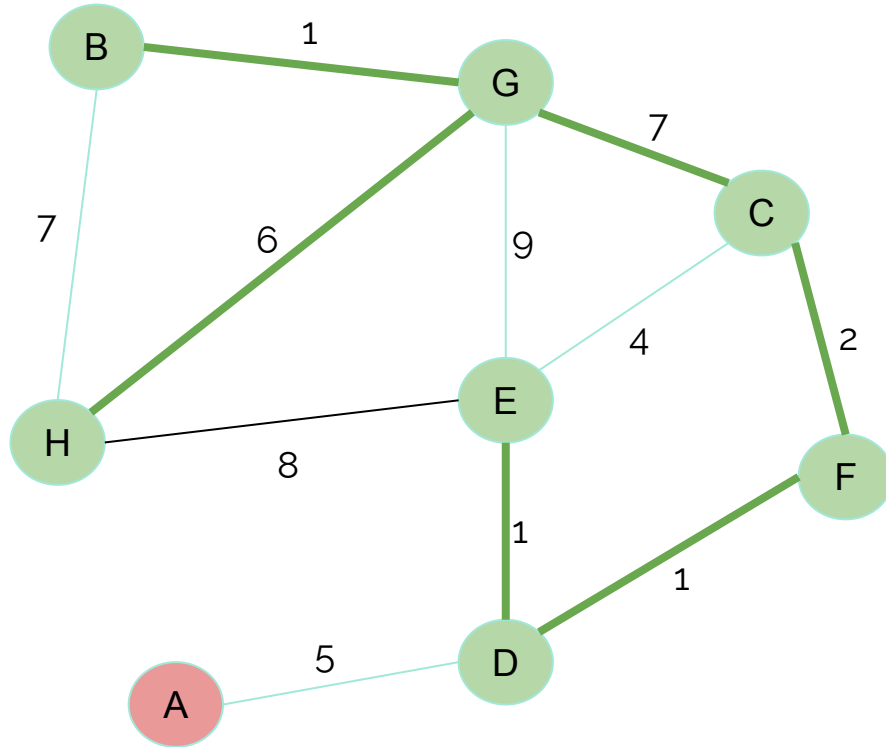
$R = \{ H, G, B, C, F, D \}$

$R' = \{ A, E \}$

— Aristas candidatas

— Aristas en MST

Ejemplo:



$V = H$

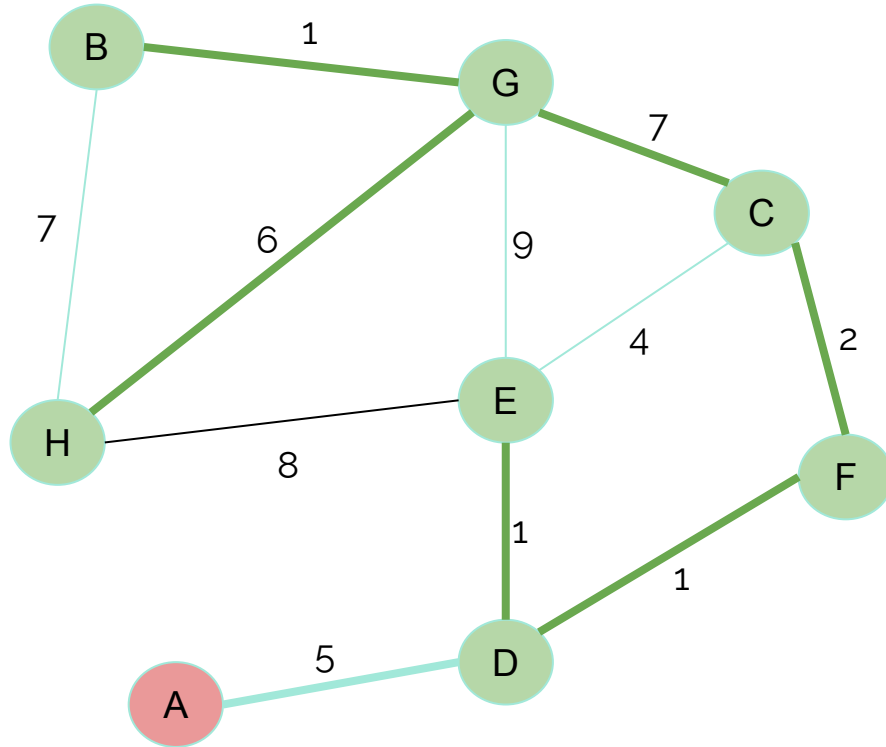
$R = \{ H, G, B, C, F, D, E \}$

$R' = \{ A \}$

— Aristas candidatas

— Aristas en MST

Ejemplo:



$V = H$

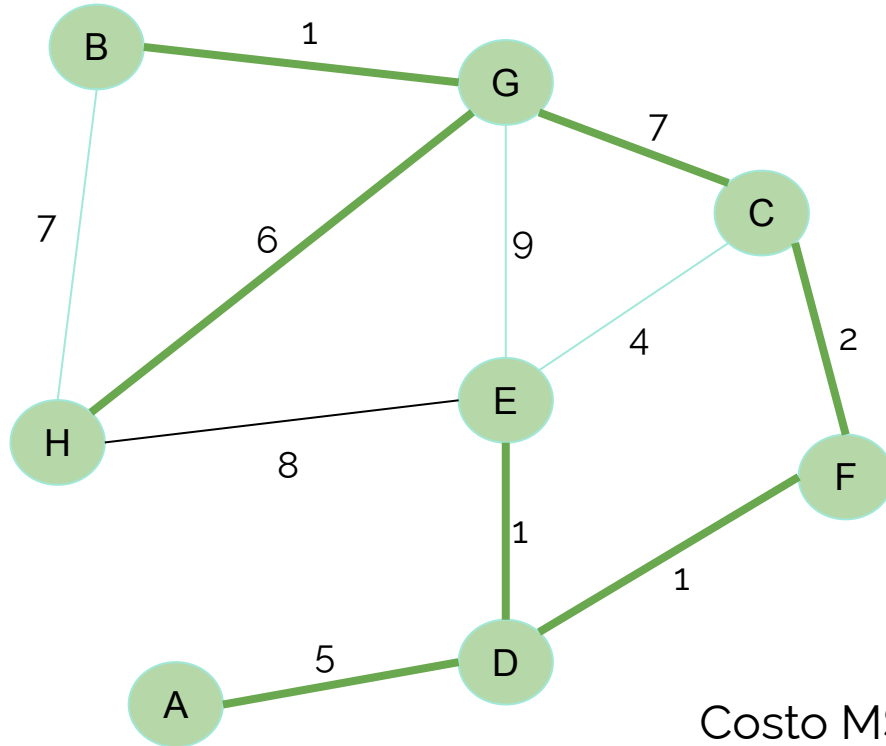
$R = \{ H, G, B, C, F, D, E \}$

$R' = \{ A \}$

 Aristas candidatas

 Aristas en MST

Ejemplo:



$V = H$

$R = \{ H, G, B, C, F, D, E, A \}$

$R' = \{ \}$

— Aristas candidatas

— Aristas en MST

Costo MST = $6 + 1 + 7 + 2 + 1 + 1 + 5 = 23$

Prim

Prim(Grafo G)

```
    set Q //de nodos ordenados por C[u]
```

```
    //C[u] es infity en un comienzo
```

```
    forest F
```

```
    while not Q.empty()
```

```
        u = min(Q)
```

```
        F.add(u)
```

```
        if E[u] != null
```

```
            F.add(E[u])
```

```
        for w,v in G.adjacent[u]
```

```
            C[v] = w
```

```
            E[v] = (u,v)
```

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.

I3 2023-1 P3

¿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: Defina primero como determinar la mejor conexión en cada caso.

I3 2023-1 P3

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 que es la mejor opción.

I3 2023-1 P3

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.