

# Algoritmo de Kruskal y conjuntos disjuntos

Clase 22

IIC 2133 - Sección 2

Prof. Mario Droguett

# Sumario

**Introducción**

Algoritmo de Kruskal

Conjuntos disjuntos

Cierre

# Componentes fuertemente conectadas

## Definición

Sea  $G$  un grafo dirigido. Una **componente fuertemente conectada (CFC)** es un conjunto maximal de nodos  $C \subseteq V(G)$  tal que dados  $u, v \in C$ , existe un camino dirigido desde  $u$  hasta  $v$

## Proposición

Si  $G$  es cíclico y los nodos de  $B \subseteq V(G)$  forman un ciclo, entonces existe una componente fuertemente conectada  $C$  tal que  $B \subseteq C$

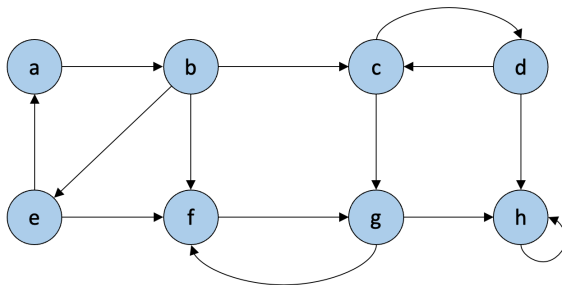
Los nodos de un ciclo pertenecen a la misma CFC

## Proposición

Un grafo  $G$  acíclico tiene 0 componentes fuertemente conectadas

# Componentes fuertemente conectadas

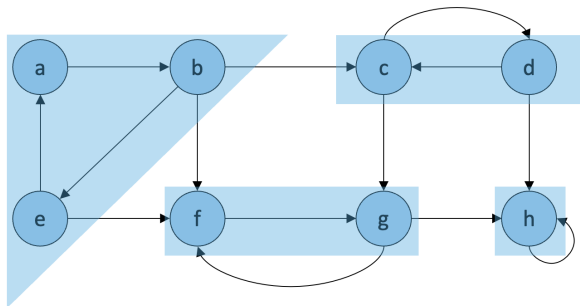
Consideremos el siguiente grafo dirigido cíclico



¿Cuáles son las componentes fuertemente conectadas de  $G$ ?

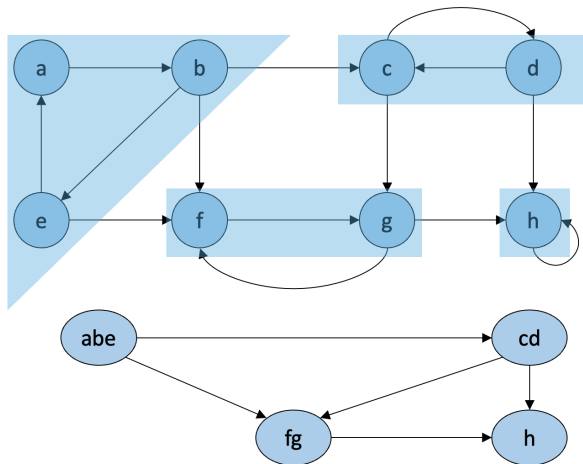
# Componentes fuertemente conectadas

Existen 4 CFC's en el grafo anterior



Notemos que es necesario poder *ir y volver* dentro de una CFC

# Componentes fuertemente conectadas



Cada componente tiene un **representante** que combina sus nodos

# Grafo transpuesto

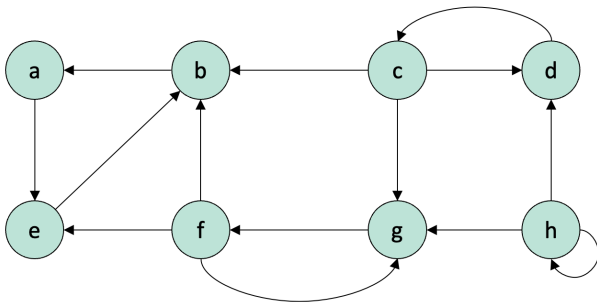
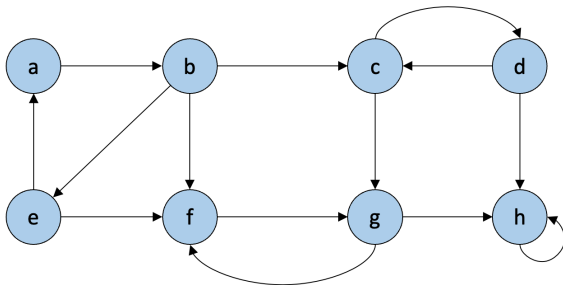
Para proponer un algoritmo, necesitamos un grafo nuevo

## Definición

Sea  $G$  un grafo dirigido. Decimos que  $G^T$  es el grafo **transpuesto** de  $G$  si

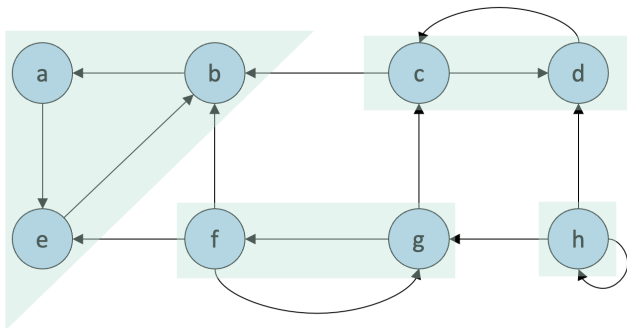
- $V(G) = V(G^T)$
- $\forall u, v \in V(G). (u, v) \in E(G) \rightarrow (v, u) \in E(G^T)$

El transpuesto se obtiene invirtiendo todas las aristas de  $G$





# Grafo transpuesto

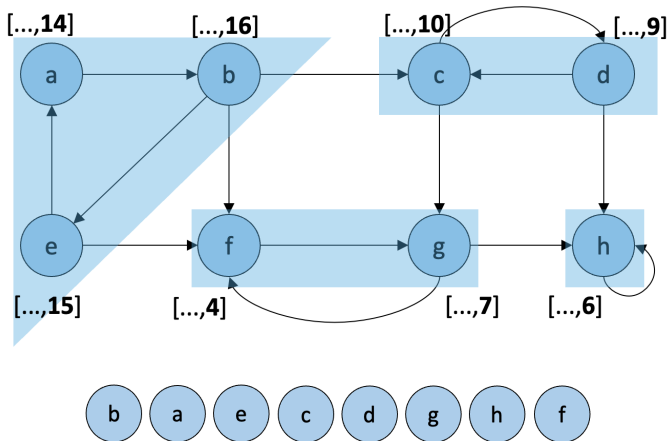


## Proposición

Los grafos  $G$  y  $G^T$  tienen las mismas componentes fuertemente conectadas

# Hacia un algoritmo para determinar las CFC

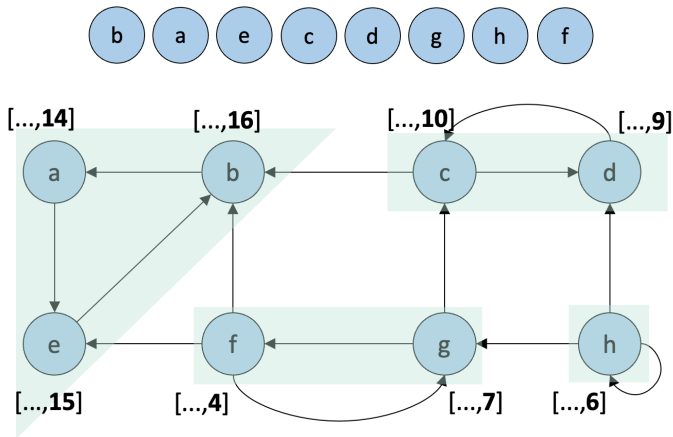
Construimos un orden de los nodos de  $G$  según tiempos de término



**¡Ojo!** Esto no es un orden topológico porque  $G$  es cíclico

# Hacia un algoritmo para determinar las CFC

Recorremos el grafo **transpuesto** partiendo según el orden anterior



Al transponer, no es posible ir de **b** a **c** porque están en componentes diferentes

# Algoritmo de Kosaraju

**input** : grafo  $G$

Kosaraju( $G$ ):

```
1   $L \leftarrow \text{TopSort}(G)$ 
2  for  $u \in L$  :
3      Assign( $G, u, u$ )
```

**input** : grafo  $G$ , nodo  $u \in V(G)$ , nodo representante  $r$

Assign( $G, u, r$ ):

```
1  if  $u.rep = \emptyset$  :
2       $u.rep \leftarrow r$ 
3      for  $v \in N_{GT}(u)$  :
4          Assign( $G, v, r$ )
```

No olvidar: no podemos interpretar  $L$  como orden topológico.  
Es un orden que se construye de la misma forma

# Algoritmo de Kosaraju

El algoritmo de Kosaraju se basa en las propiedades del siguiente grafo

## Definición

Dado un grafo  $G$  dirigido, sean  $C_1, \dots, C_k$  sus componentes fuertemente conectadas. Se define el **grafo de componentes**  $G^{CFC}$  según

- $V(G^{CFC}) = \{C_1, \dots, C_k\}$
- Si  $(u, v) \in E(G)$  y  $u \in C_i, v \in C_j$ , entonces  $(C_i, C_j) \in E(G^{CFC})$

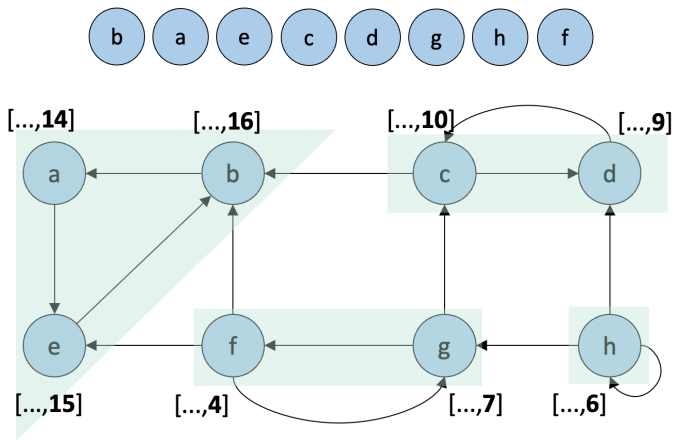
## Teorema

El grafo de componentes  $G^{CFC}$  es un grafo dirigido acíclico

## Corolario

El grafo de componentes  $G^{CFC}$  tiene un orden topológico

# Hacia un algoritmo para determinar las CFC



La forma en que recorremos las componentes nos da su orden topológico  
 $(bae)(cd)(gf)(h)$

# Árboles de cobertura mínimos

## Definición

Dado un grafo no dirigido  $G$ , un subgrafo  $T \subseteq G$  se dice un **árbol de cobertura mínimo** o **MST** de  $G$  si

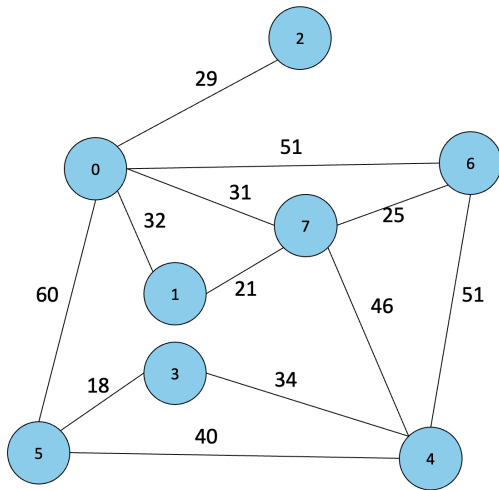
1.  $T$  es un árbol
2.  $V(T) = V(G)$
3. No existe otro MST  $T'$  para  $G$  con menor costo total

Es decir, si  $T$  es MST de  $G$ ,

1. no tiene ciclos
2. es una **cobertura** de los nodos de  $G$
3. tiene costo mínimo

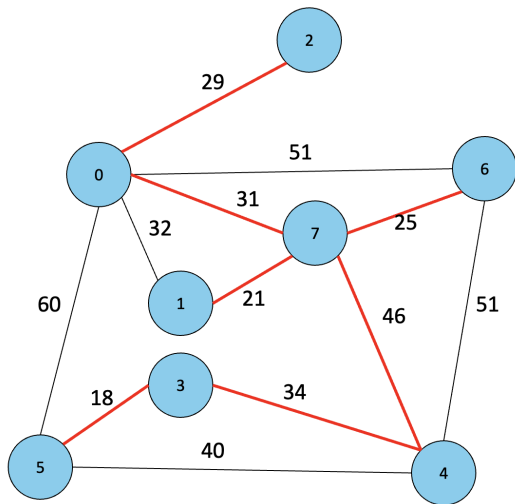
¿Puede haber más de un MST diferente para  $G$ ?

## Árboles de cobertura mínimos





## Árboles de cobertura mínimos



# Árboles de cobertura mínimos

Los MST están presentes en la solución de múltiples problemas de conectividad

- Redes de distribución eléctrica
- Redes telefónicas
- Comunicaciones, computacionales, tráfico aéreo. . .
- Incluso redes biológicas, químicas y físicas

# Árboles de cobertura mínimos

Para atacar el problema algorítmicamente, dado  $G$  no dirigido

- Llamamos **corte** a una **partición**  $(V_1, V_2)$  de  $V(G)$

$$V_1, V_2 \neq \emptyset, \quad V_1 \cup V_2 = V(G), \quad V_1 \cap V_2 = \emptyset$$

- Diremos que una arista **cruza el corte** si uno de sus extremos está en  $V_1$  y el otro en  $V_2$

¿Qué podemos afirmar respecto a los MST y las aristas que cruzan un corte dado?

# Árboles de cobertura mínimos

Si  $T$  es un MST de  $G$ , y  $P = (V_1, V_2)$  es un corte de  $G$

- Sabemos que al menos una arista que cruza  $P$  está incluida en  $T$
- De lo contrario, no habría camino entre nodos de  $V_1$  y  $V_2$
- Por lo tanto:  $T$  no sería de cobertura

Si  $P = (V_1, V_2)$  es un corte de  $G$

- Sabemos que cada arista de corte tiene un costo asociado
- La arista más barata **siempre** se incluye en **algún** MST

Usaremos estas ideas para construir un MST desde cero

# Sumario

Introducción

**Algoritmo de Kruskal**

Conjuntos disjuntos

Cierre

# Algoritmo de Kruskal

La idea detrás del **algoritmo de Kruskal** es crear un bosque que va convergiendo en un único árbol

Para un grafo  $G = (V, E)$ , 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

¿Es necesario revisar **todas** las aristas de  $E$ ?

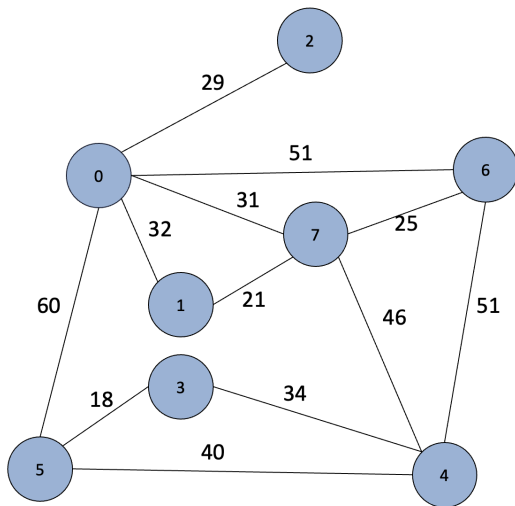
# Algoritmo de Kruskal

Kruskal( $G$ ):

```
1   $E \leftarrow E$  ordenada por costo, de menor a mayor
2   $T \leftarrow$  lista vacía
3  for  $e \in E$  :
4      if Agregar  $e$  a  $T$  no forma ciclo :
5           $T \leftarrow T \cup \{e\}$ 
6  return  $T$ 
```

¿Este algoritmo usa cortes de manera implícita?

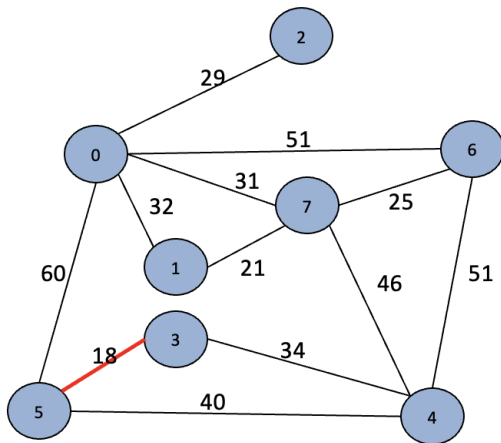
# Algoritmo de Kruskal



Al inicio, tenemos un bosque de  $|V|$  árboles sin aristas

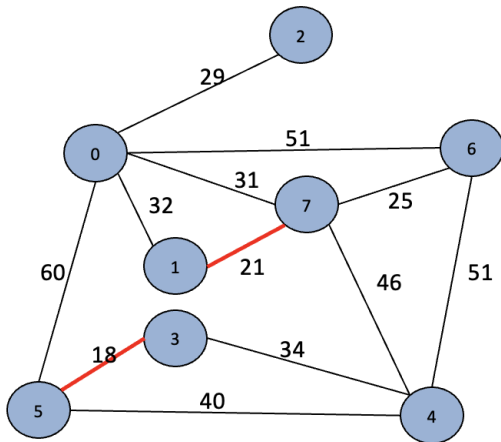


# Algoritmo de Kruskal

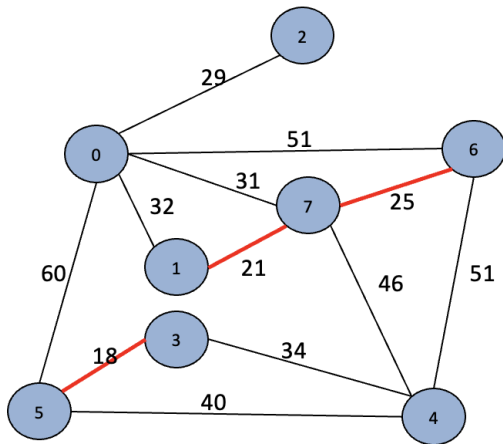


Reducimos la cantidad de árboles en 1 con cada arista añadida

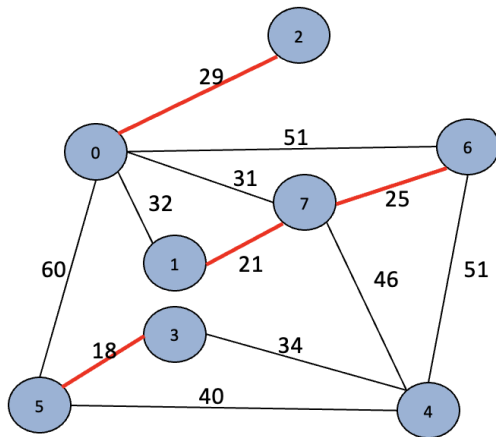
# Algoritmo de Kruskal



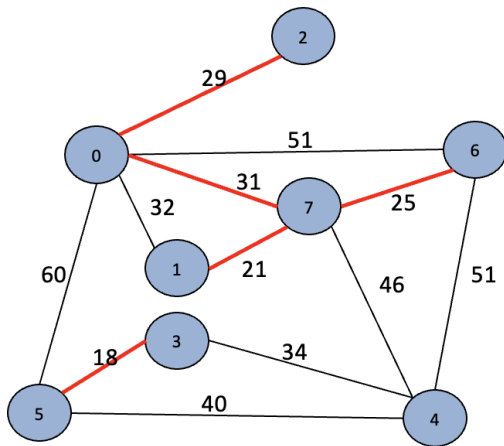
# Algoritmo de Kruskal



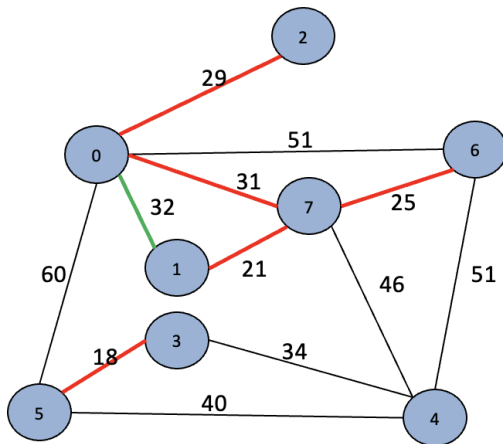
# Algoritmo de Kruskal



# Algoritmo de Kruskal



# Algoritmo de Kruskal



# Algoritmo de Kruskal: una conexión con cortes

Dada una arista  $(u, v)$  que corresponde chequear

- Podemos considerar un corte dado por

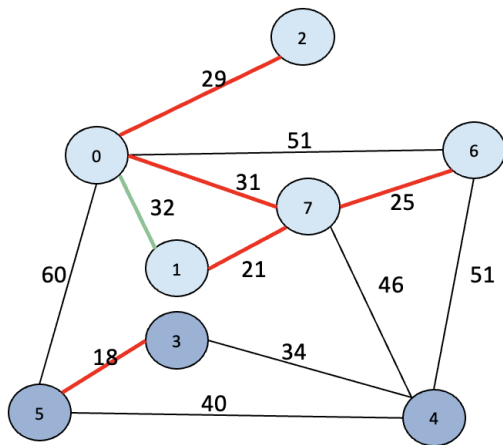
$$V_1 = \{w \mid w \text{ está conectado con } u \text{ con aristas de } T\}, \quad V_2 = V - V_1$$

- Y otro corte dado por

$$V_1 = \{w \mid w \text{ está conectado con } v \text{ con aristas de } T\}, \quad V_2 = V - V_1$$

Agregamos la arista si es de corte para estos dos cortes

# Algoritmo de Kruskal

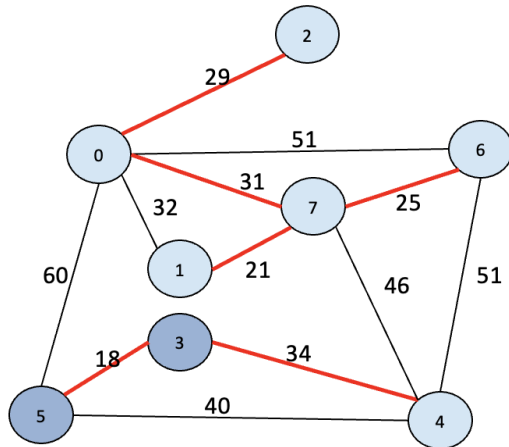


La arista  $(0,1)$  no es de corte para ninguno de los cortes (ambos cortes son iguales porque 0 y 1 ya están conectados entre sí):

$$(\{0, 1, 2, 6, 7\}, \{3, 4, 5\})$$



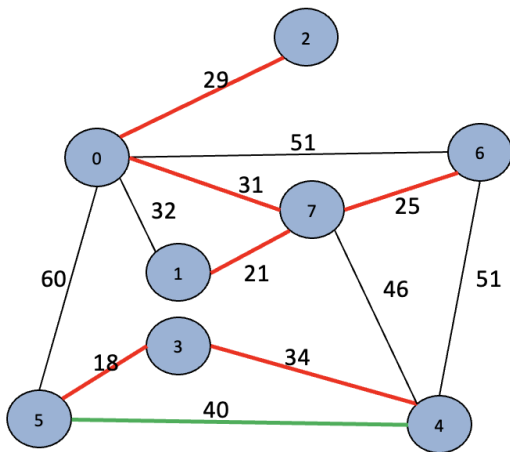
# Algoritmo de Kruskal



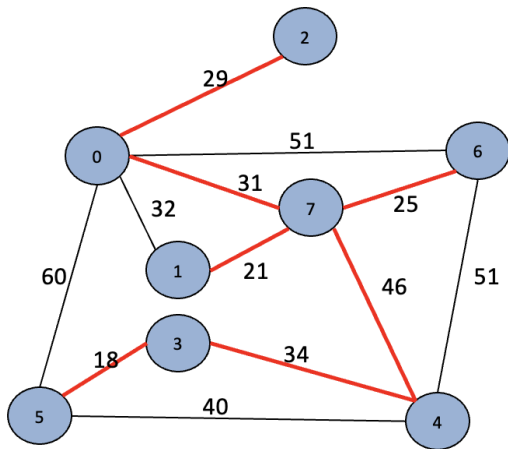
La arista (3,4) es de corte para los cortes:

$$(\{3,5\}, \{0,1,2,4,6,7\}) \quad (\{4\}, \{0,1,2,3,5,6,7\})$$

# Algoritmo de Kruskal

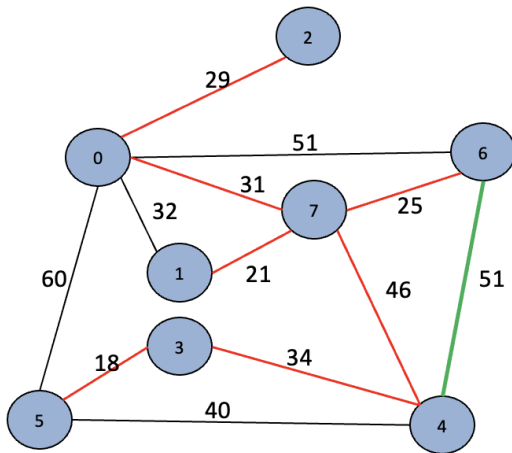


# Algoritmo de Kruskal



En este punto,  $|E| = |V| - 1$  y  $T$  es un árbol

# Algoritmo de Kruskal



Cualquier arista adicional formaría un **ciclo**

# Algoritmo de Kruskal

Kruskal( $G$ ):

```
1   $E \leftarrow E$  ordenada por costo, de menor a mayor
2   $T \leftarrow$  lista vacía
3  for  $e \in E$  :
4      if Agregar  $e$  a  $T$  no forma ciclo :
5           $T \leftarrow T \cup \{e\}$ 
6  return  $T$ 
```

¿Cómo revisamos eficientemente que  $e$  no forma un ciclo en  $T$ ?

# Algoritmo de Kruskal: conjuntos

Dada una arista  $(u, v)$  podemos considerar los conjuntos

- Nodos conectados con  $u$  en  $T$

$$V_u = \{w \mid w \text{ está conectado con } u \text{ con aristas de } T\}$$

- Nodos conectados con  $v$  en  $T$

$$V_v = \{w \mid w \text{ está conectado con } v \text{ con aristas de } T\}$$

La arista forma un **ciclo** en  $T$  si, y solo si,  $V_u = V_v$

Notemos que los árboles del bosque  $T$  forman **conjuntos disjuntos**

¿Cómo modelar eficientemente estas estructuras?

# Sumario

Introducción

Algoritmo de Kruskal

**Conjuntos disjuntos**

Cierre

# Conjuntos disjuntos

## Definición

Una colección de **conjuntos disjuntos**  $\{S_1, \dots, S_n\}$  es una EDD que permite

- Identificar a qué conjunto **pertenece** un elemento
- **Unir** conjuntos  $S_i, S_j$  formando un nuevo conjunto

Para atacar la representación de los conjuntos usaremos un **representante**

- Un elemento cualquiera de  $S$  que denotamos por  $rep(S)$
- La consulta de representante debe ser consistente si no hay cambios en  $S$ : entrega el mismo elemento
- Cada elemento de  $S$  tiene referencia a  $rep(S)$

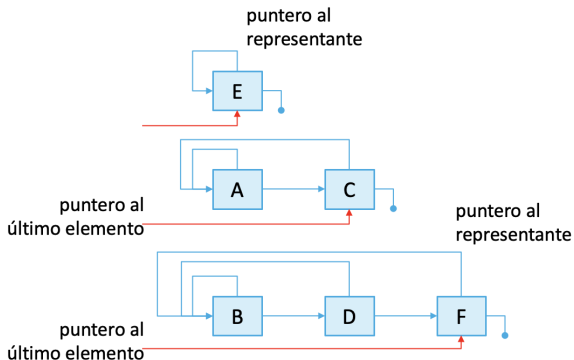
Dos conjuntos  $S$  y  $T$  son iguales si y solo si  $rep(S) = rep(T)$



# Conjunto disjuntos: listas ligadas

Una primera representación puede ser con listas ligadas

Para los conjuntos  $\{E\}$ ,  $\{A, C\}$ ,  $\{B, D, F\}$



¿Cuál es la complejidad de las operaciones de búsqueda y unión?

# Setting

Supondremos que las operaciones son de la forma

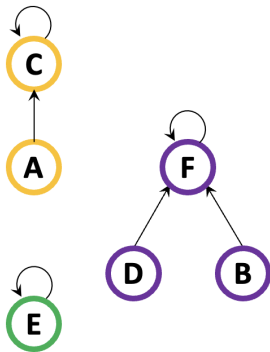
- $\text{Find}(A)$ : entrega el conjunto al que pertenece  $A$
- $\text{Union}(A, B)$ : entrega un conjunto resultante de unir los conjuntos de  $A$  y  $B$

Además, supondremos que en un estado inicial, tenemos  $n$  conjuntos con **un solo elemento cada uno**. Un conjunto con un elemento se llama **singleton**

Conectando estos conjuntos podremos resolver Kruskal

# Conjuntos como árboles

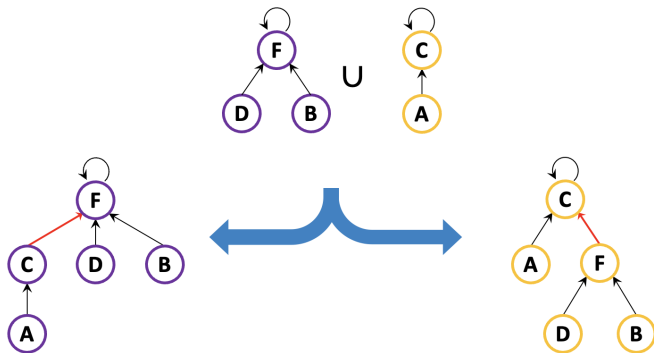
Una primera forma de representar los conjuntos es usar **árboles** donde los caminos llevan al **representante**



¿Cómo se implementan las operaciones Union y Find?

# Conjuntos como árboles

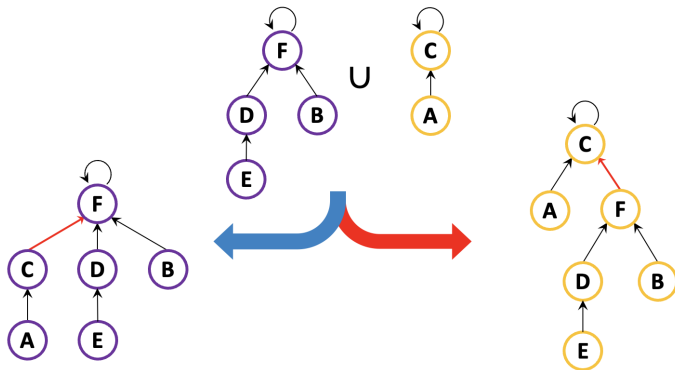
La operación **Union** corresponde a cambiar el autoloop del representante de un conjunto:  $\mathcal{O}(1)$



¿Cuál de las dos opciones escogemos como resultado?

# Conjuntos como árboles

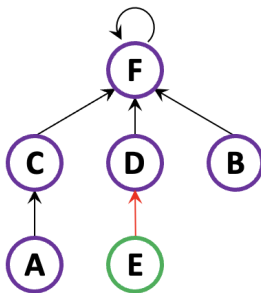
La operación Find requiere recorrer punteros, por lo que las dos opciones no son equivalentes



Hay que unir el árbol más corto al más largo

# Conjuntos como árboles

La operación Find requiere recorrer punteros

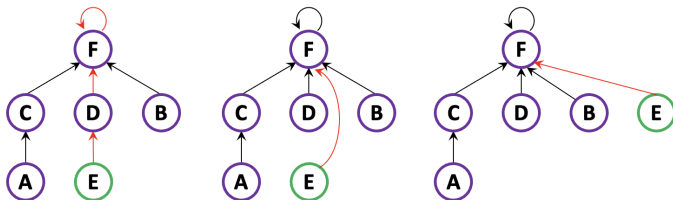


$$\text{Find}(E) = \text{Find}(D) = \text{Find}(F) = F$$

¿Podríamos mejorar estructuralmente?

# Conjuntos como árboles

Podemos modificar punteros para simplificar las rutas



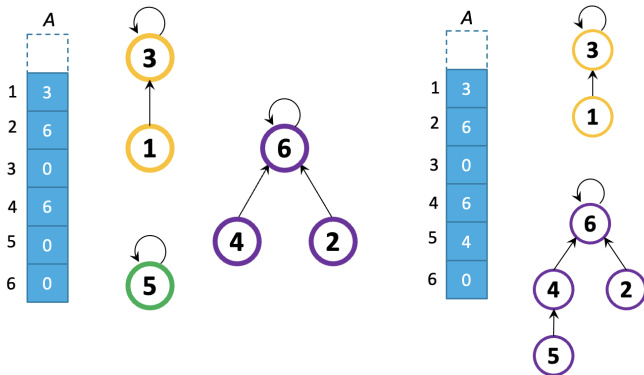
$$\text{Find}(E) == F$$

¿Cómo almacenamos los conjuntos?

# Conjuntos como árboles

Podemos almacenar los árboles en un único arreglo de referencias:

$A[k]$  es el padre de  $k$





# Complejidad de Find

El costo de Find depende de qué elemento se busca

- Se puede demostrar que para un conjunto de  $n$  elementos, con rutas simplificadas, Find toma tiempo  $\mathcal{O}(\alpha(n))$
- $\alpha(n)$  es una función *interesante* que tiene buenas propiedades
- En particular, crece **extremadamente lento**

$$\alpha(n) = 4, \quad 2048 \leq n \leq 16^{512}$$

En cualquier aplicación práctica, podemos considerar que  $\alpha(n) \in \mathcal{O}(1)$

# Algoritmo de Kruskal y conjuntos

Ahora que tenemos una implementación de conjuntos

- En Kruskal comenzamos con un conjunto para cada nodo
- Verificamos aristas viendo si sus extremos están **en el mismo conjunto**
- Si no lo están, las agregamos y **unimos** los conjuntos

# Algoritmo de Kruskal y conjuntos

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  $T$ 
```

¿Qué complejidad tiene este algoritmo?

# Complejidad

Usando la implementación de conjuntos disjuntos

- Ordenar las aristas  $\mathcal{O}(E \log(E))$
- Construir  $V$  conjuntos singleton  $\mathcal{O}(V)$
- Unir conjuntos  $V - 1$  veces  $\mathcal{O}(V)$
- Buscar  $2E$  veces  $\mathcal{O}(E\alpha(V)) = \mathcal{O}(E)$

En total

$$\mathcal{O}(E \log(E) + V + E) = \mathcal{O}(E \log(E))$$

y como  $E \leq V^2$ , tenemos que  $\log(E) \in \mathcal{O}(\log(V))$

Kruskal tiene complejidad  $\mathcal{O}(E \log(V))$

# Sumario

Introducción

Algoritmo de Kruskal

Conjuntos disjuntos

**Cierre**

# Objetivos de la clase

- ☐ Comprender una estrategia codiciosa para escoger aristas de un MST
- ☐ Comprender el algoritmo de Kruskal
- ☐ Relacionar MST con el problema de conjuntos disjuntos
- ☐ Justificar la complejidad de Kruskal en base a operaciones de conjuntos