Tema 4-4

Árboles generadores



UNIVERSIDAD DE CHILE

Arturo Merino



Recuerdo/Plan

Recuerdo y plan

- **Recuerdo:**

Recuerdo y plan

- **Recuerdo:**
  - Conexidad

Recuerdo y plan

- **Recuerdo:**
  - Conexidad
  - Árboles

Recuerdo y plan

- **Recuerdo:**
  - Conexidad
  - Árboles

- **Plan:**

Recuerdo y plan

- **Recuerdo:**
  - Conexidad
  - Árboles

- **Plan:**
  - Árboles generadores de costo mínimo

Recuerdo y plan

- **Recuerdo:**
  - Conexidad
  - Árboles

- **Plan:**
  - Árboles generadores de costo mínimo
  - Prim y Kruskal.

Árboles generadores

Árboles generadores

- Árbol $\leftarrow$ certificado *óptimo* de conexidad.

Árboles generadores

- Árbol $\leftarrow$ certificado *óptimo* de conexidad.

Def.

Sea G un grafo simple.

Árboles generadores

- Árbol $\leftarrow$ certificado *óptimo* de conexidad.

Def.

Sea G un grafo simple.

Un *árbol generador* de G es un subgrafo T que contiene todos los vértices de G y es un árbol.

Árboles generadores

- Árbol $\leftarrow$ certificado *óptimo* de conexidad.

Def.

Sea G un grafo simple.

Un *árbol generador* de G es un subgrafo T que contiene todos los vértices de G y es un árbol.

- Existe si y sólo si G es conexo.

Árboles generadores

- Árbol $\leftarrow$ certificado *óptimo* de conexidad.

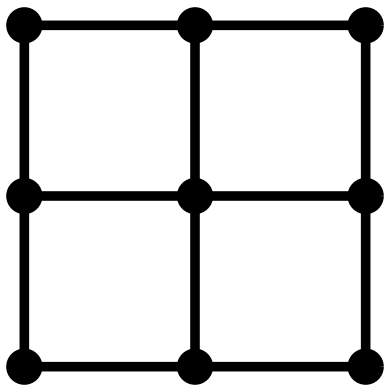
Def.

Sea G un grafo simple.

Un *árbol generador* de G es un subgrafo T que contiene todos los vértices de G y es un árbol.

- Existe si y sólo si G es conexo.

- **Ej:**



Árboles generadores

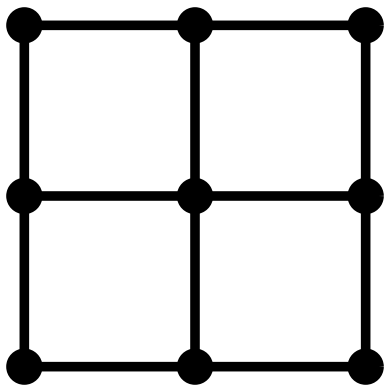
- Árbol $\leftarrow$ certificado *óptimo* de conexidad.

Def.

Sea G un grafo simple.

Un *árbol generador* de G es un subgrafo T que contiene todos los vértices de G y es un árbol.

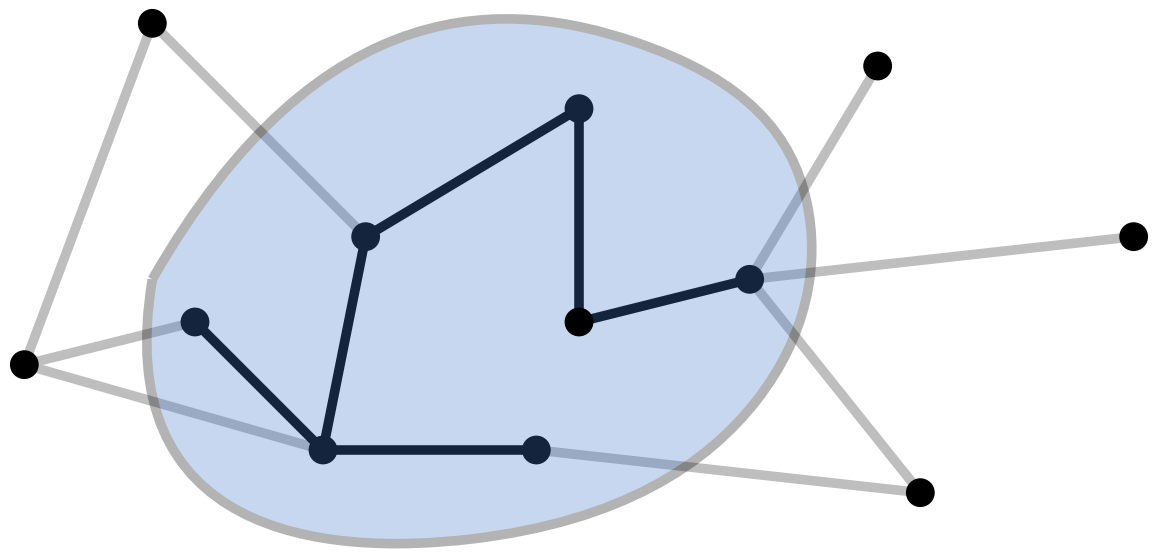
- Existe si y sólo si G es conexo.
- **Ej:**



- ¿Cómo encontrarlos/construirlos?

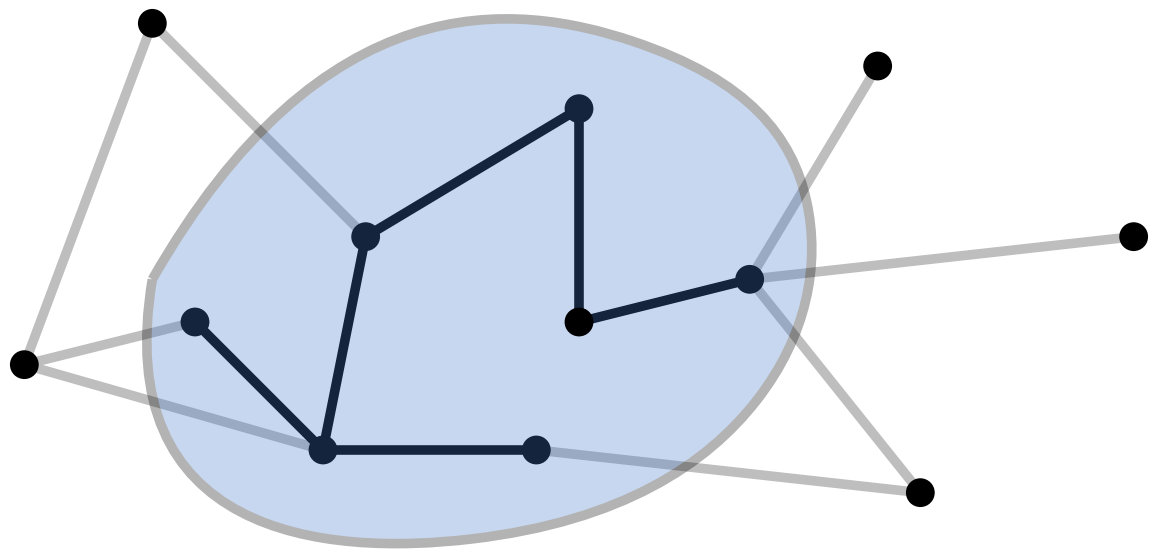
Árbol incremental

- **Idea:** ir de manera incremental.



Árbol incremental

- **Idea:** ir de manera incremental.

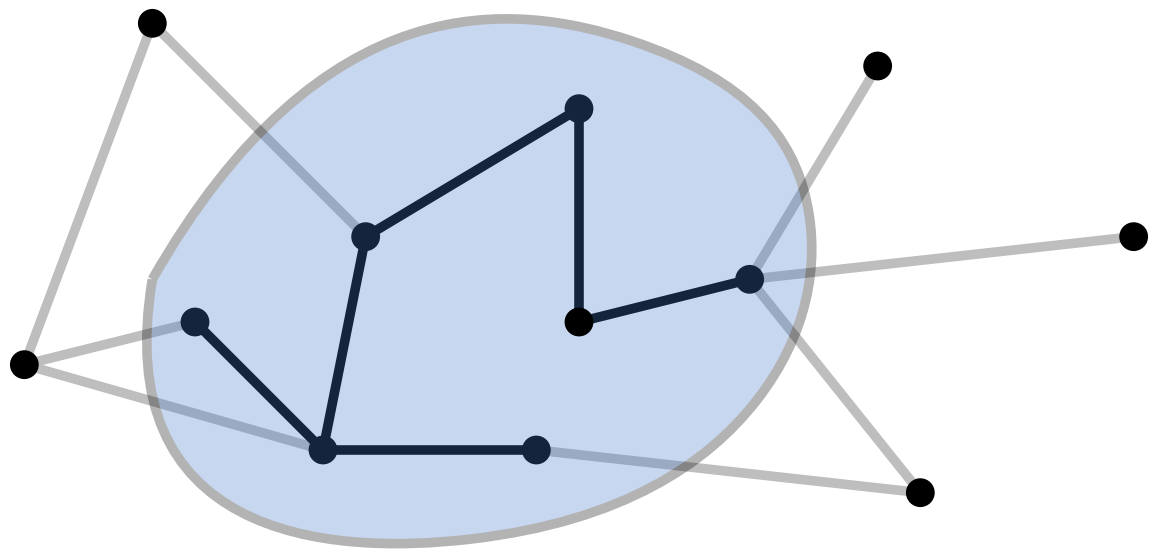


Lem.

Sean $G = (V, E)$ un grafo simple y (U, T) un subgrafo árbol de G.

Árbol incremental

- **Idea:** ir de manera incremental.



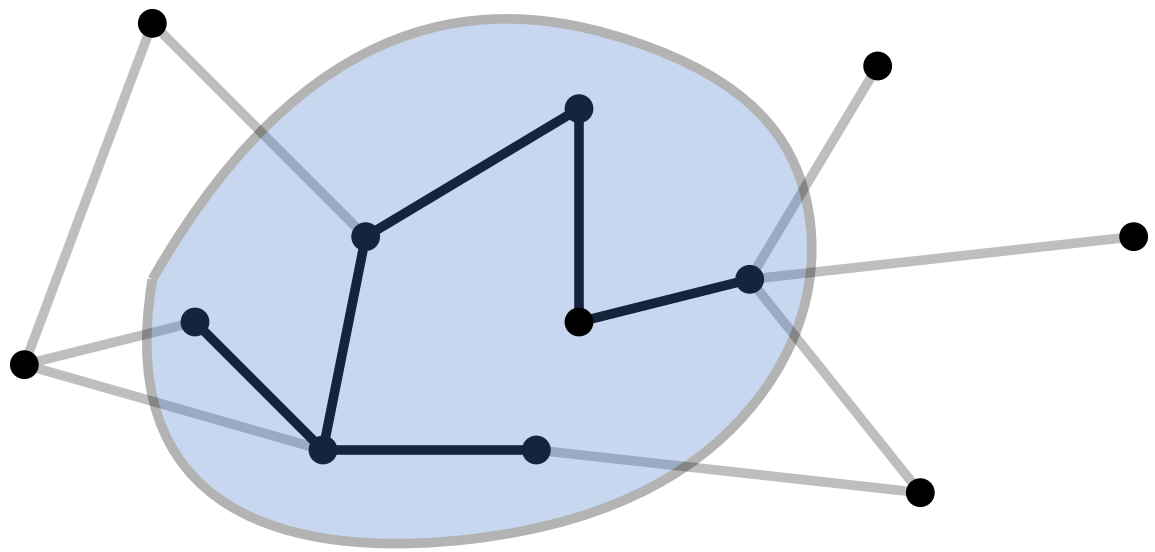
Lem.

Sean $G = (V, E)$ un grafo simple y (U, T) un subgrafo árbol de G.

Si $uv \in E$ conecta $u \in U$ y $v \notin U$, entonces $(U \cup \{v\}, T \cup \{uv\})$ es un árbol.

Árbol incremental

- **Idea:** ir de manera incremental.



Lem.

Sean $G = (V, E)$ un grafo simple y (U, T) un subgrafo árbol de G.

Si $uv \in E$ conecta $u \in U$ y $v \notin U$, entonces $(U \cup \{v\}, T \cup \{uv\})$ es un árbol.

- ¡Nos da un algoritmo!

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

Elegir una arista uv con esa propiedad.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

Elegir una arista uv con esa propiedad.

$U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

    Elegir una arista uv con esa propiedad.

    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

    Elegir una arista uv con esa propiedad.

    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

- La regla de elección puede ser cualquiera.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

    Elegir una arista uv con esa propiedad.

    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

- La regla de elección puede ser cualquiera.

Teo.

Sean G un grafo simple y $r \in V(G)$.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

Elegir una arista uv con esa propiedad.

$U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

- La regla de elección puede ser cualquiera.

Teo.

Sean G un grafo simple y $r \in V(G)$.

La exploración devuelve un árbol generador de la componente conexa de r.

Algoritmo de exploración

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while existe $uv \in E$ con $u \in U$ y $v \notin U$:

Elegir una arista uv con esa propiedad.

$U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

- La regla de elección puede ser cualquiera.

Teo.

Sean G un grafo simple y $r \in V(G)$.

La exploración devuelve un árbol generador de la componente conexa de r.

- ¿Qué hay que probar?

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Si H es conexo y no existe $uv \in E$ con $u \in U$ y $v \notin U$, entonces H es una componente conexa de G.

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Si H es conexo y no existe $uv \in E$ con $u \in U$ y $v \notin U$, entonces H es una componente conexa de G.

- **Dem:**

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Si H es conexo y no existe $uv \in E$ con $u \in U$ y $v \notin U$, entonces H es una componente conexa de G.

- **Dem:**

- **Concluimos:** correctitud para cualquier regla.

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Si H es conexo y no existe $uv \in E$ con $u \in U$ y $v \notin U$, entonces H es una componente conexa de G.

- **Dem:**

- **Concluimos:** correctitud para cualquier regla.
- ¿Qué pasa para reglas específicas?

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Si H es conexo y no existe $uv \in E$ con $u \in U$ y $v \notin U$, entonces H es una componente conexa de G.

- **Dem:**

- **Concluimos:** correctitud para cualquier regla.
- ¿Qué pasa para reglas específicas?

Def.

- *Tiempo de llegada* $\tau(v)$: paso en que v entra en U; $\tau(r) = 0$.

Correctitud de la exploración

Lem.

Sean $G = (V, E)$ un grafo simple y H un subgrafo de G.

Si H es conexo y no existe $uv \in E$ con $u \in U$ y $v \notin U$, entonces H es una componente conexa de G.

- **Dem:**

- **Concluimos:** correctitud para cualquier regla.
- ¿Qué pasa para reglas específicas?

Def.

- *Tiempo de llegada* $\tau(v)$: paso en que v entra en U; $\tau(r) = 0$.
- *Nivel* de v: largo del camino simple de r a v en el árbol.

Búsqueda en anchura (BFS)

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Búsqueda en anchura (BFS)

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

Búsqueda en anchura (BFS)

Alg.

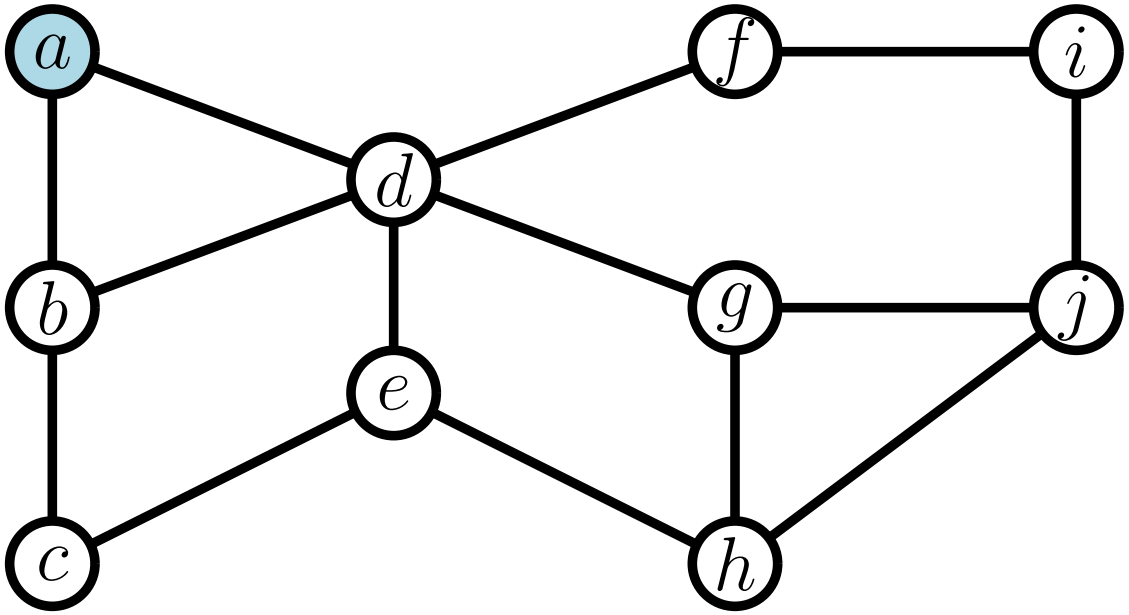
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$$U = \{a\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

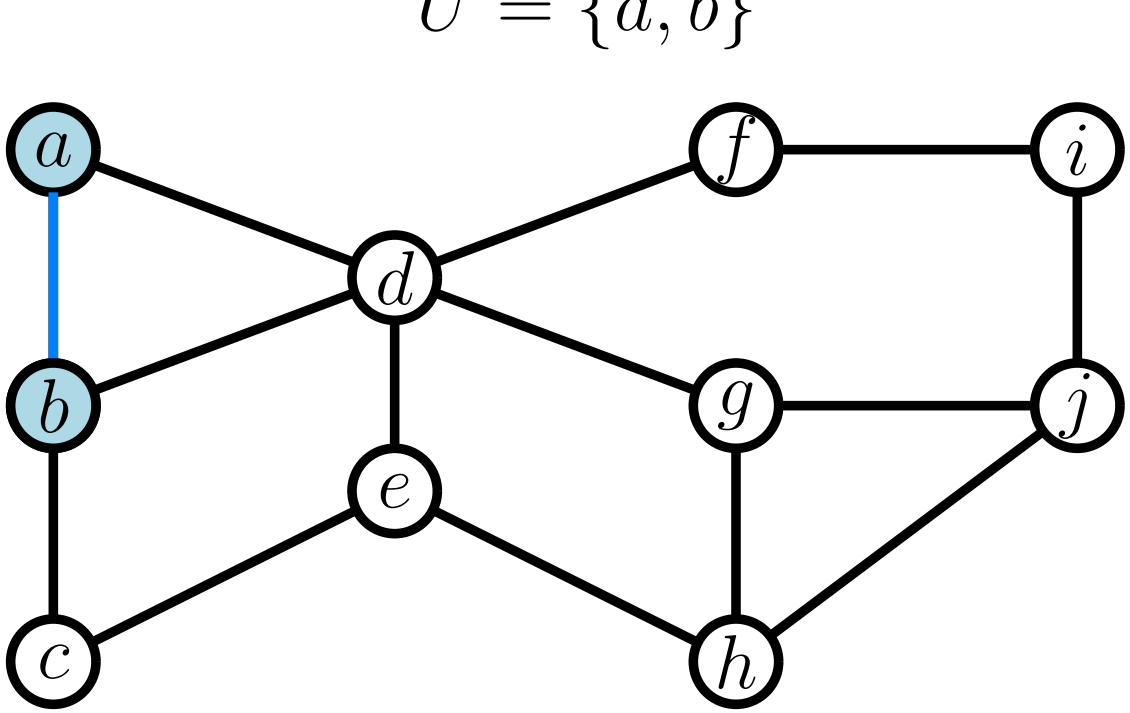
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$U = \{a, b\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

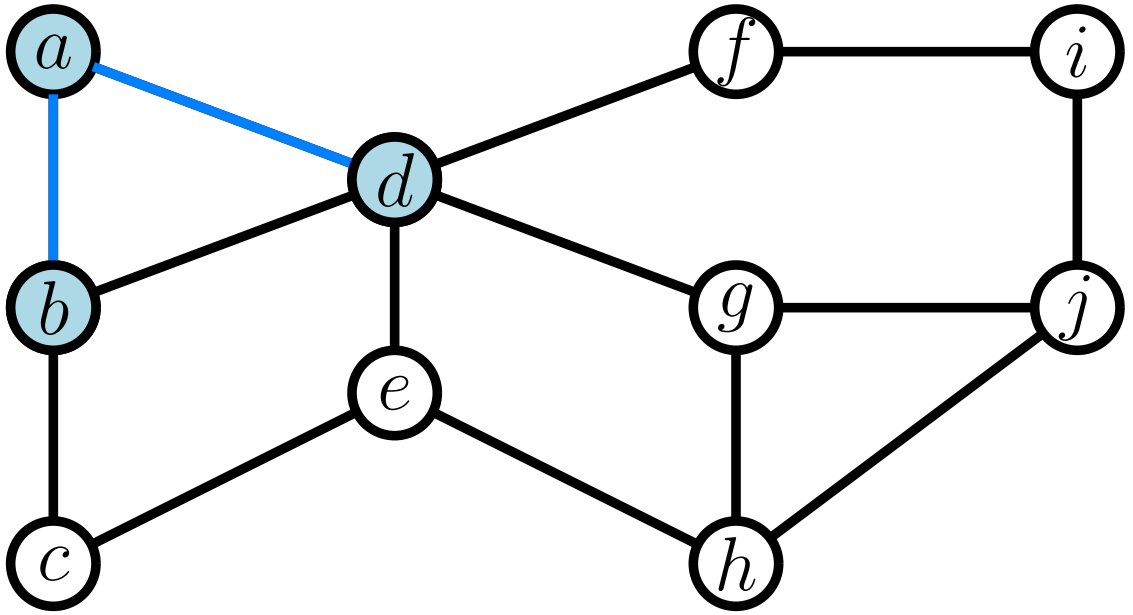
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$U = \{a, b, d\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

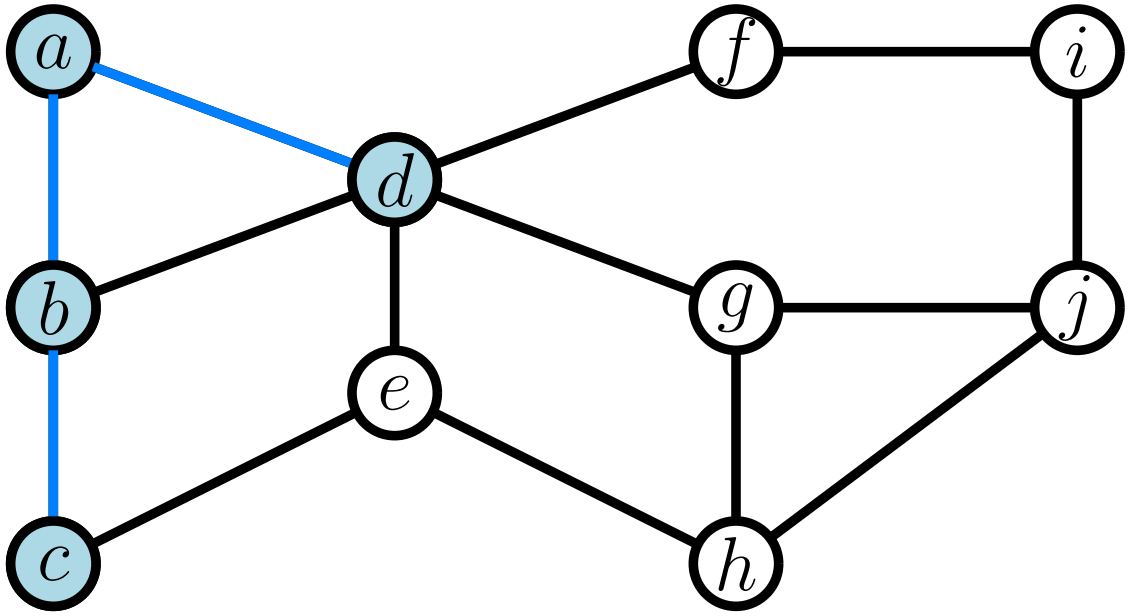
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$U = \{a, b, c, d\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

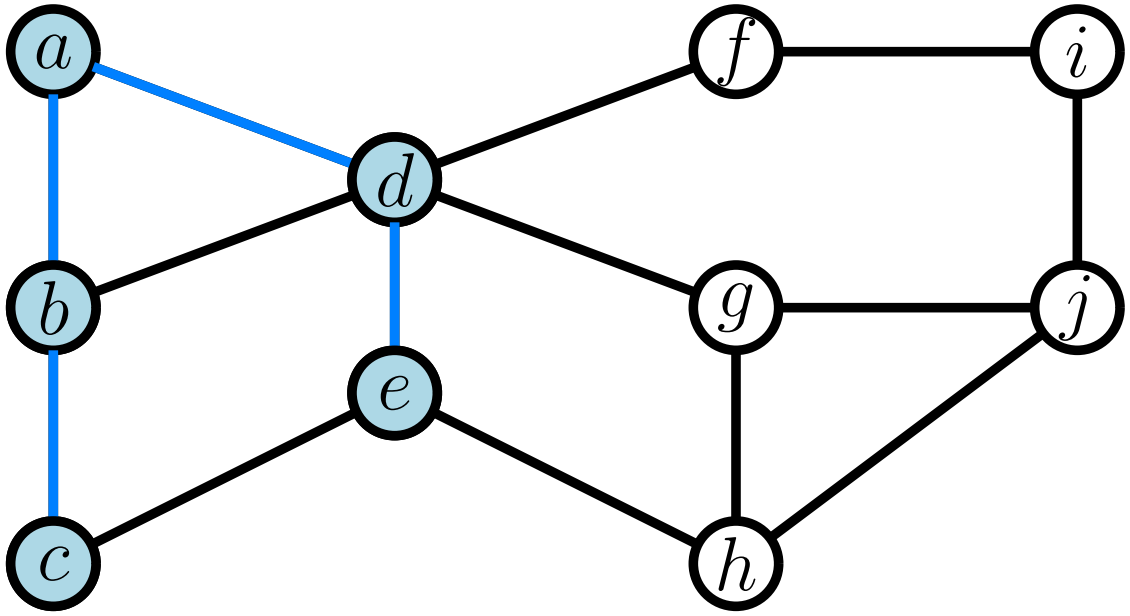
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$$U = \{a, b, c, d, e\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2
4	de	e	2

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

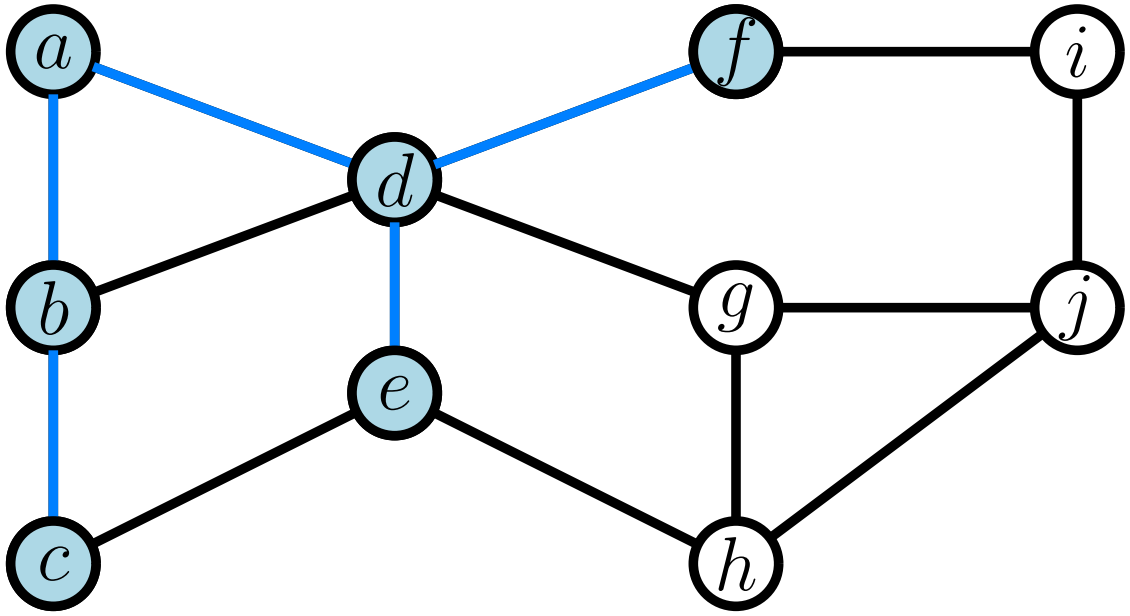
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2
4	de	e	2
5	df	f	2

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

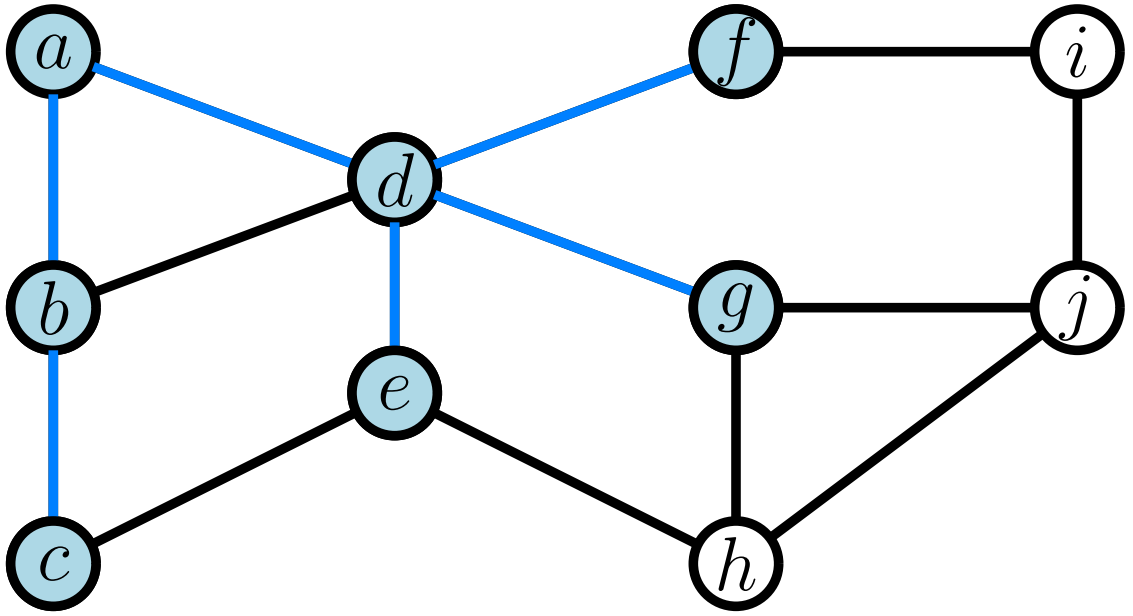
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f, g\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2
4	de	e	2
5	df	f	2
6	dg	g	2

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

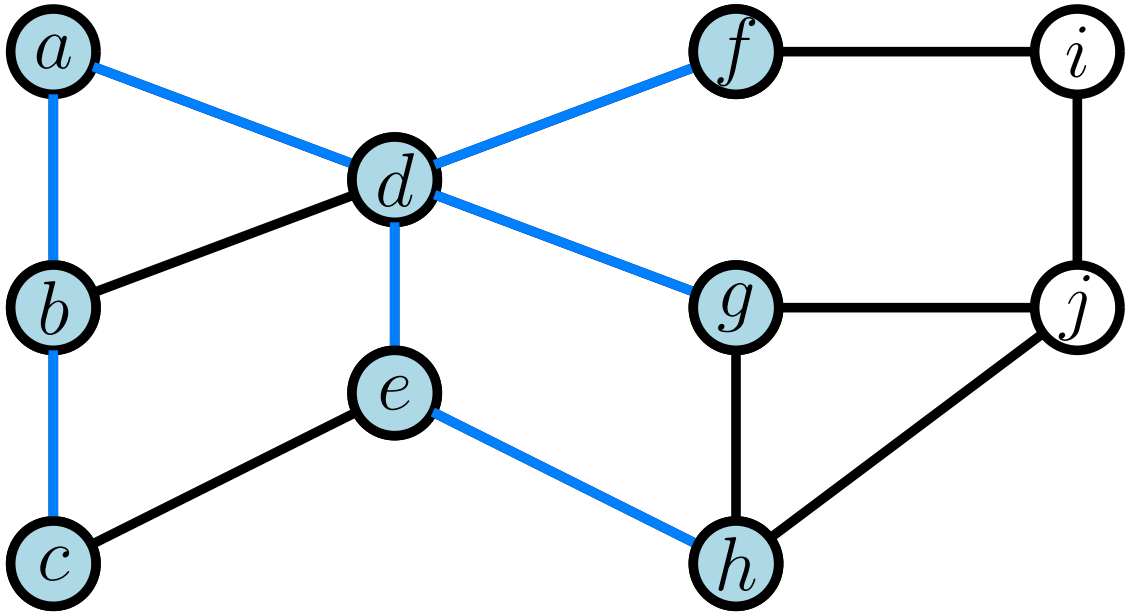
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f, g, h\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2
4	de	e	2
5	df	f	2
6	dg	g	2
7	eh	h	3

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

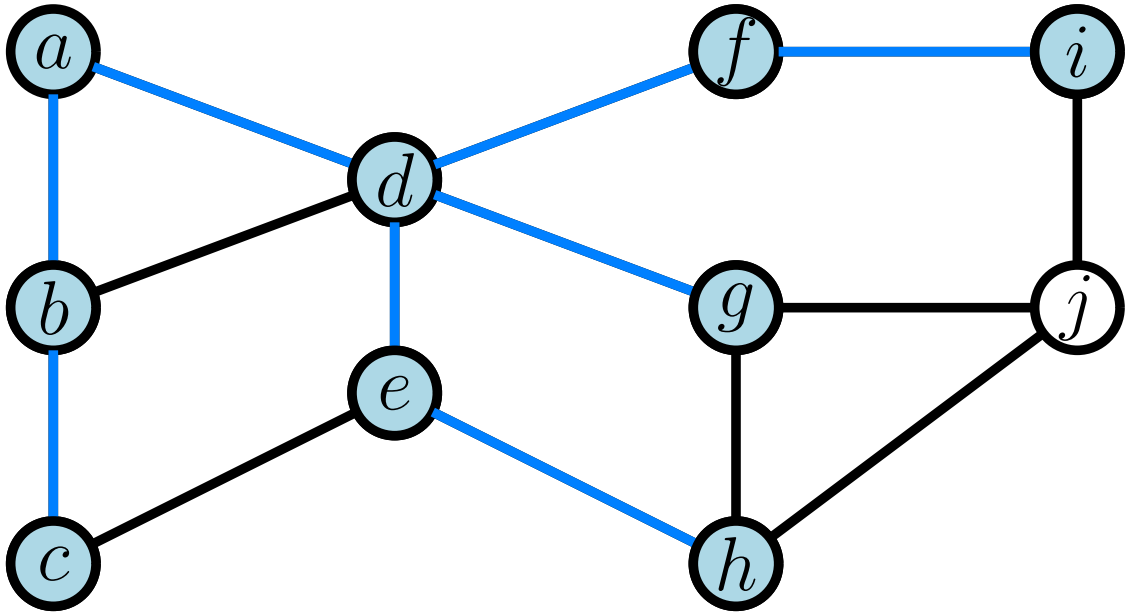
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$U = \{a, b, c, d, e, f, g, h, i\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2
4	de	e	2
5	df	f	2
6	dg	g	2
7	eh	h	3
8	fi	i	3

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Alg.

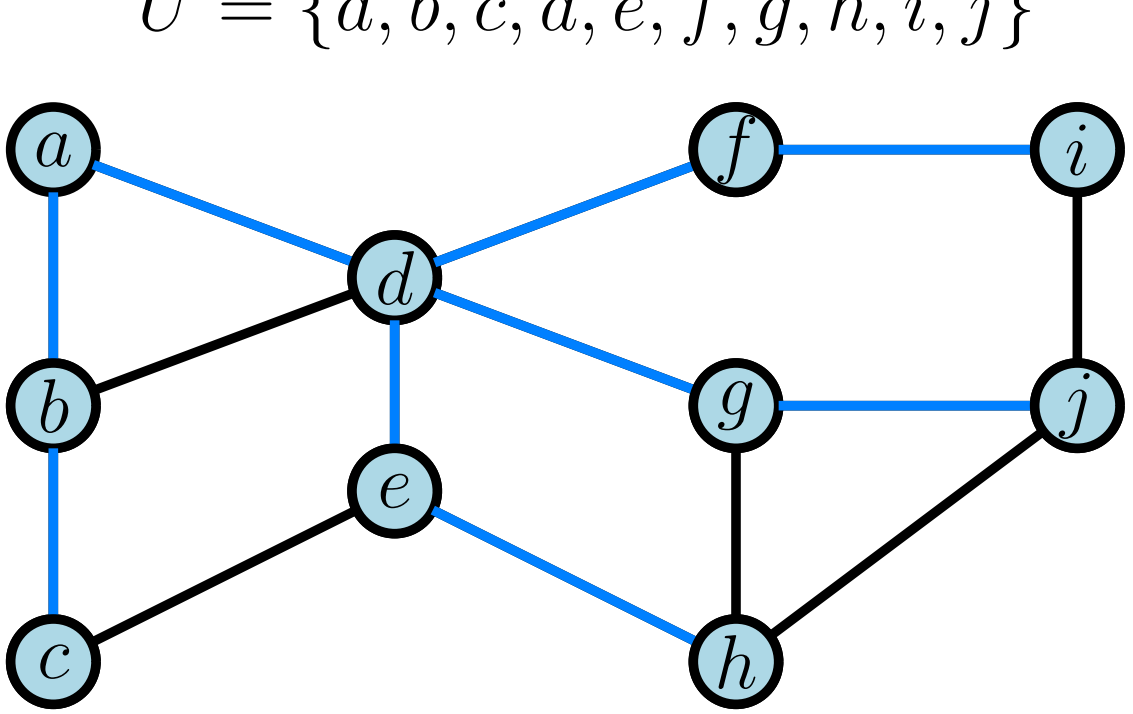
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más viejo posible.

- **Ej:**

$U = \{a, b, c, d, e, f, g, h, i, j\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	ad	d	1
3	bc	c	2
4	de	e	2
5	df	f	2
6	dg	g	2
7	eh	h	3
8	fi	i	3
9	gj	j	3

  - Implementación vía cola.

Búsqueda en anchura (BFS)

Def.

Sean G un grafo simple y conexo y $r, v \in V(G)$.

Búsqueda en anchura (BFS)

Def.

Sean G un grafo simple y conexo y $r, v \in V(G)$.

La *distancia* $d_G(r, v)$ es el mínimo largo de un camino de r a v.

Búsqueda en anchura (BFS)

Def.

Sean G un grafo simple y conexo y $r, v \in V(G)$.

La *distancia* $d_G(r, v)$ es el mínimo largo de un camino de r a v.

Prop.

Sean G un grafo simple y conexo, y $r \in V(G)$.

Búsqueda en anchura (BFS)

Def.

Sean G un grafo simple y conexo y $r, v \in V(G)$.

La *distancia* $d_G(r, v)$ es el mínimo largo de un camino de r a v.

Prop.

Sean G un grafo simple y conexo, y $r \in V(G)$.

BFS ubica cada vértice v en el nivel $d_G(r, v)$.

Búsqueda en anchura (BFS)

Def.

Sean G un grafo simple y conexo y $r, v \in V(G)$.

La *distancia* $d_G(r, v)$ es el mínimo largo de un camino de r a v.

Prop.

Sean G un grafo simple y conexo, y $r \in V(G)$.

BFS ubica cada vértice v en el nivel $d_G(r, v)$.

- **Dem:**

Búsqueda en profundidad (DFS)

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Búsqueda en profundidad (DFS)

Alg.

Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

Búsqueda en profundidad (DFS)

Alg.

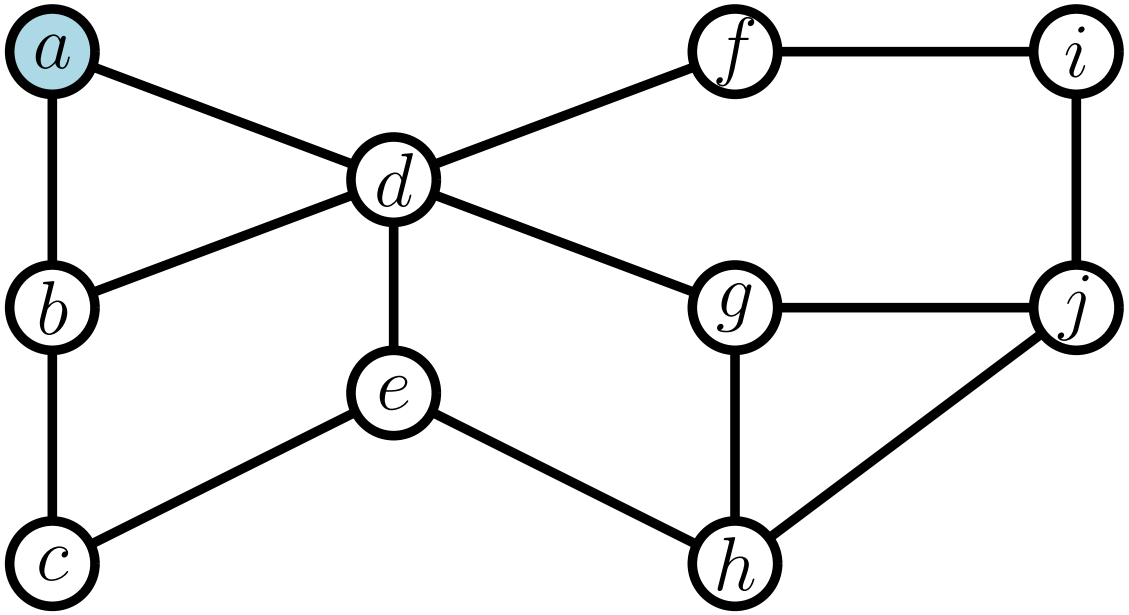
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

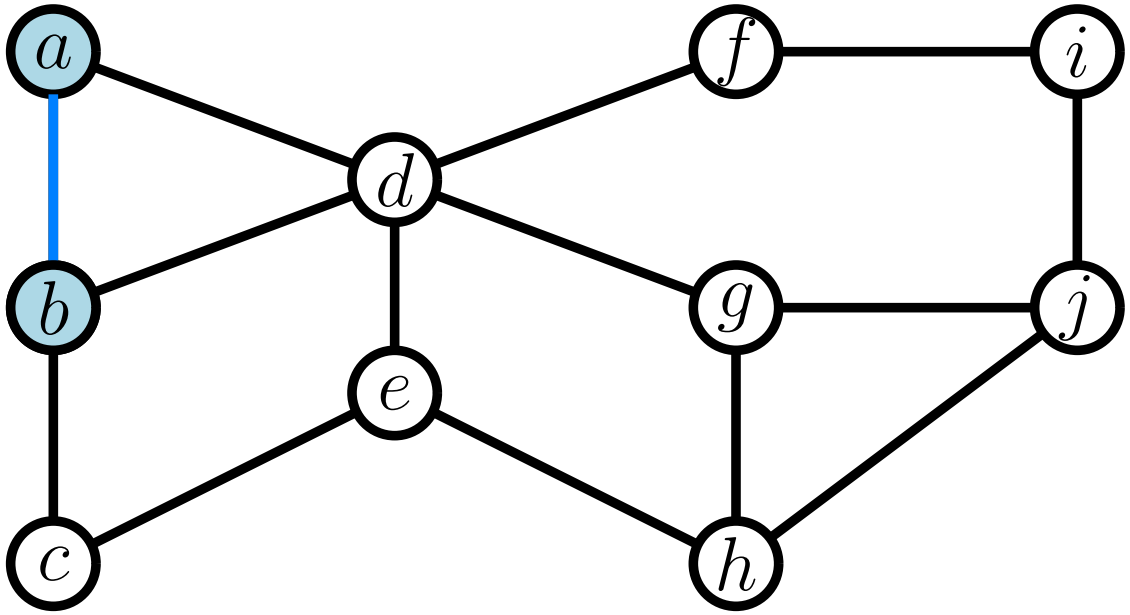
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$U = \{a, b\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

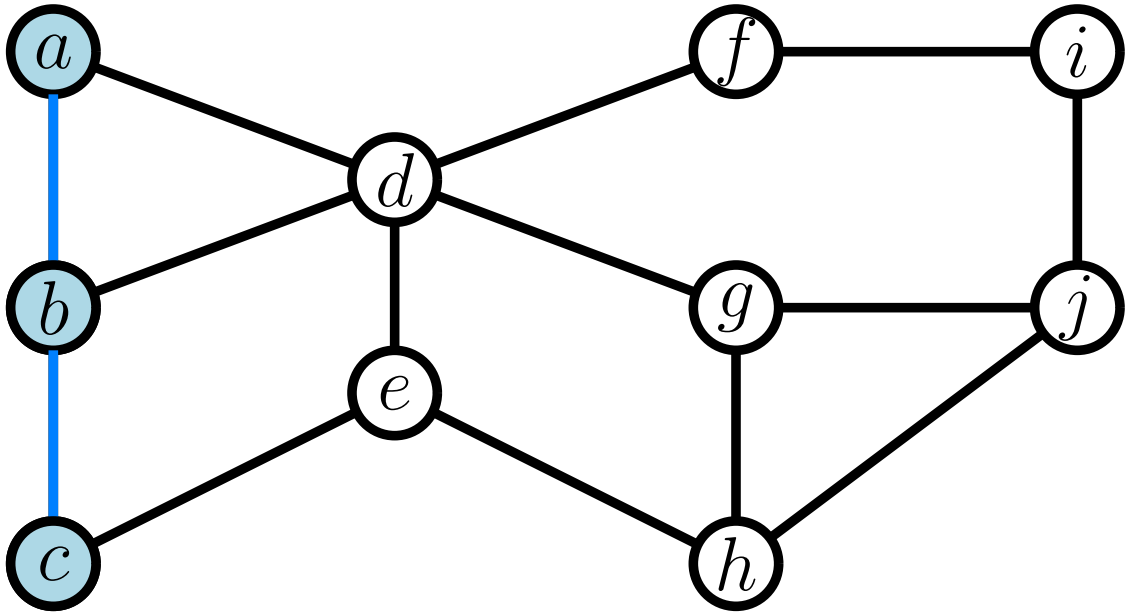
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$U = \{a, b, c\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

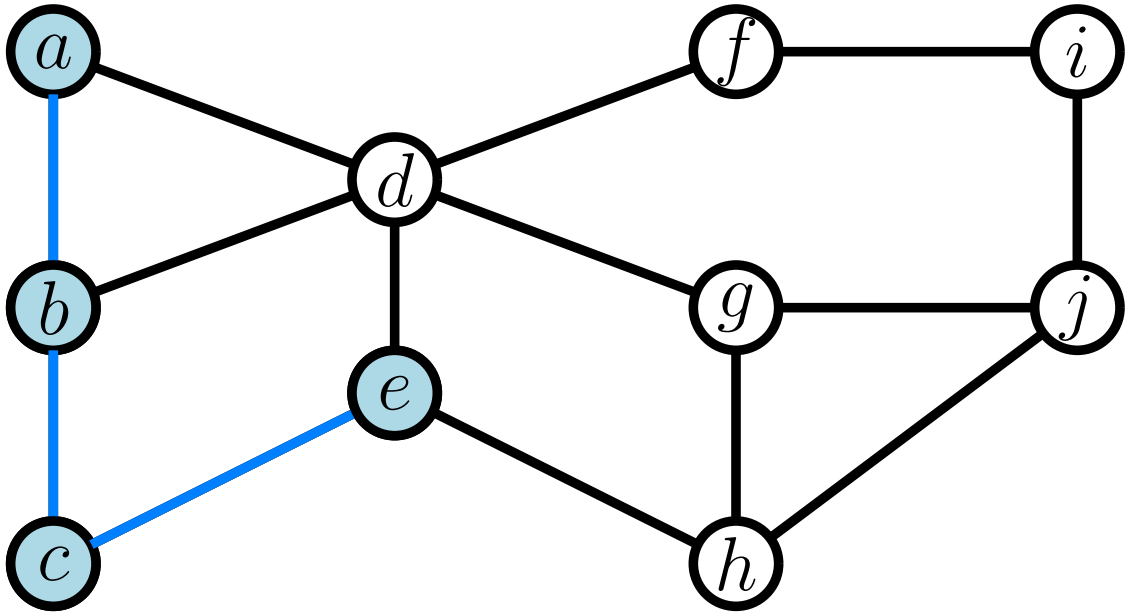
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a, b, c, e\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

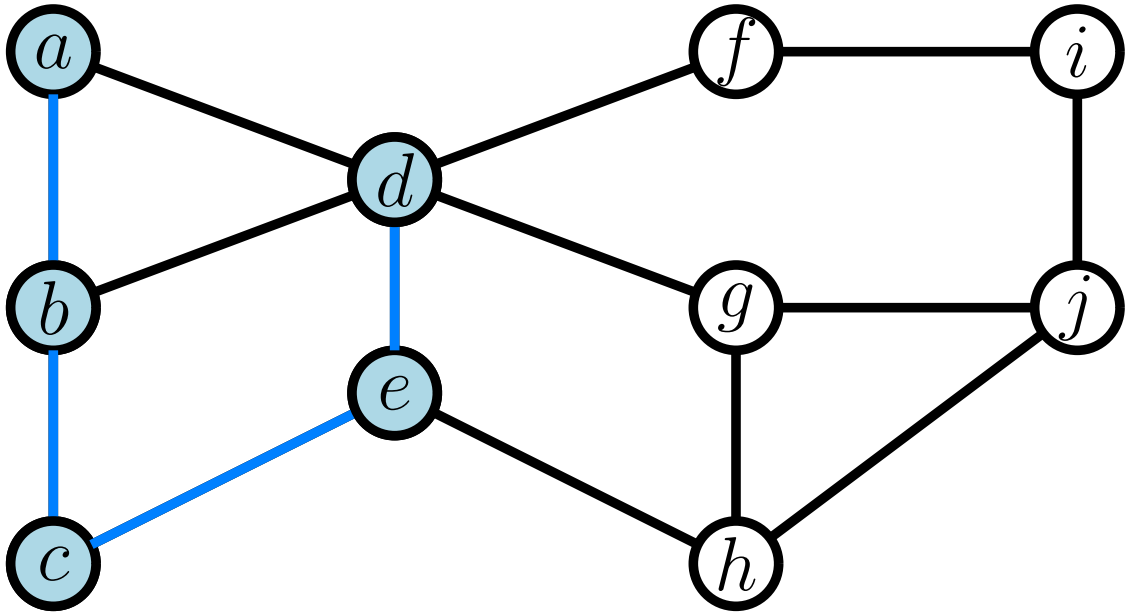
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a, b, c, d, e\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3
4	ed	d	4

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

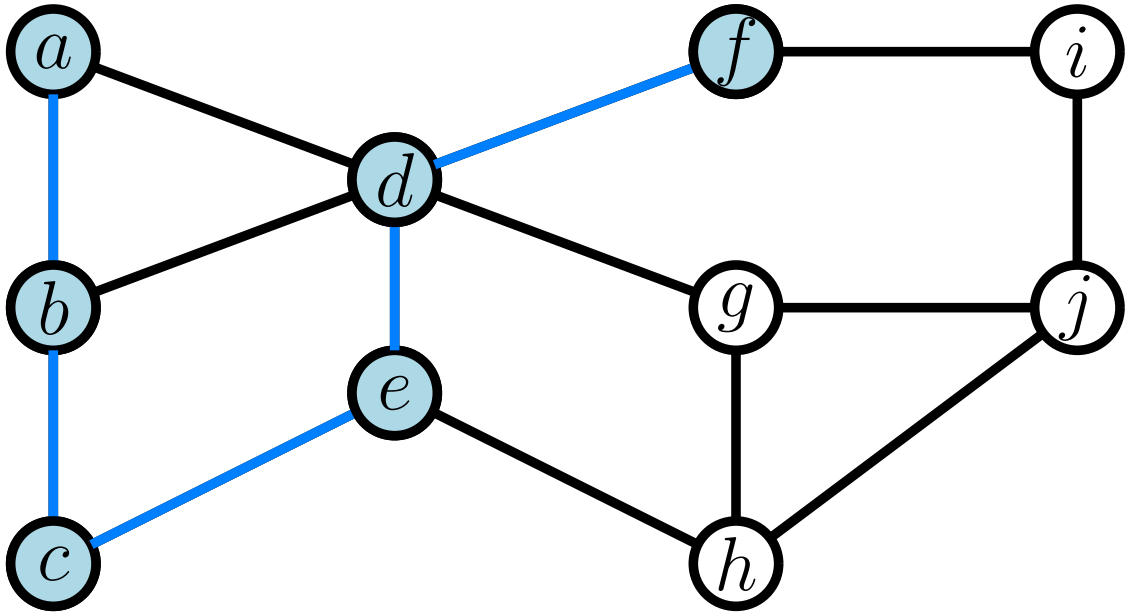
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3
4	ed	d	4
5	df	f	5

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

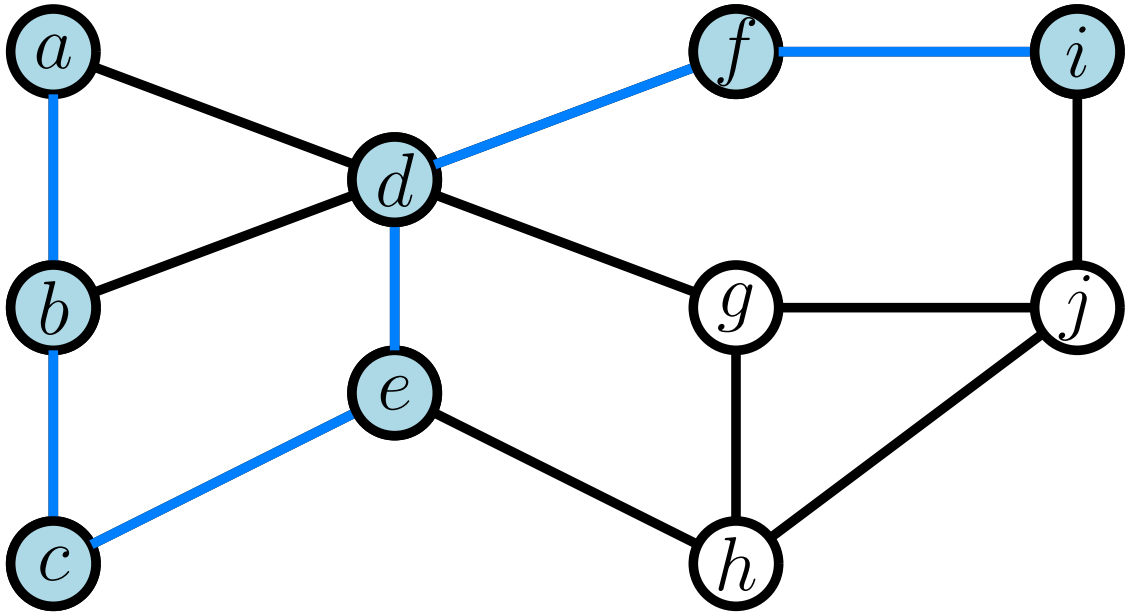
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f, i\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3
4	ed	d	4
5	df	f	5
6	fi	i	6

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

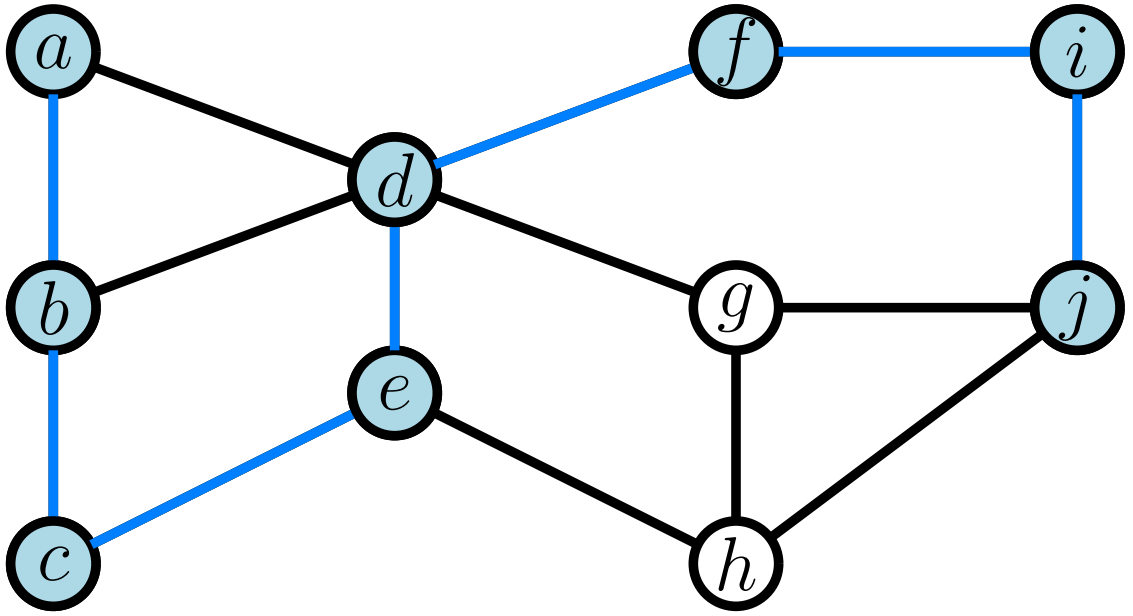
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f, i, j\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3
4	ed	d	4
5	df	f	5
6	fi	i	6
7	ij	j	7

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

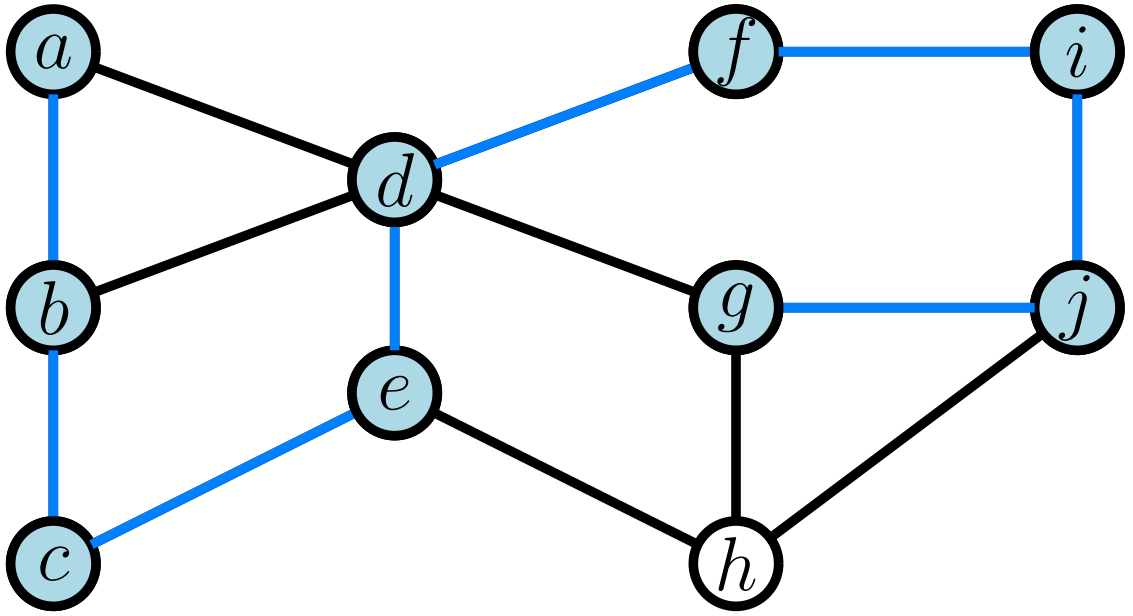
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$$U = \{a, b, c, d, e, f, g, i, j\}$$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3
4	ed	d	4
5	df	f	5
6	fi	i	6
7	ij	j	7
8	jg	g	8

  - Implementación vía pila.

Búsqueda en profundidad (DFS)

Alg.

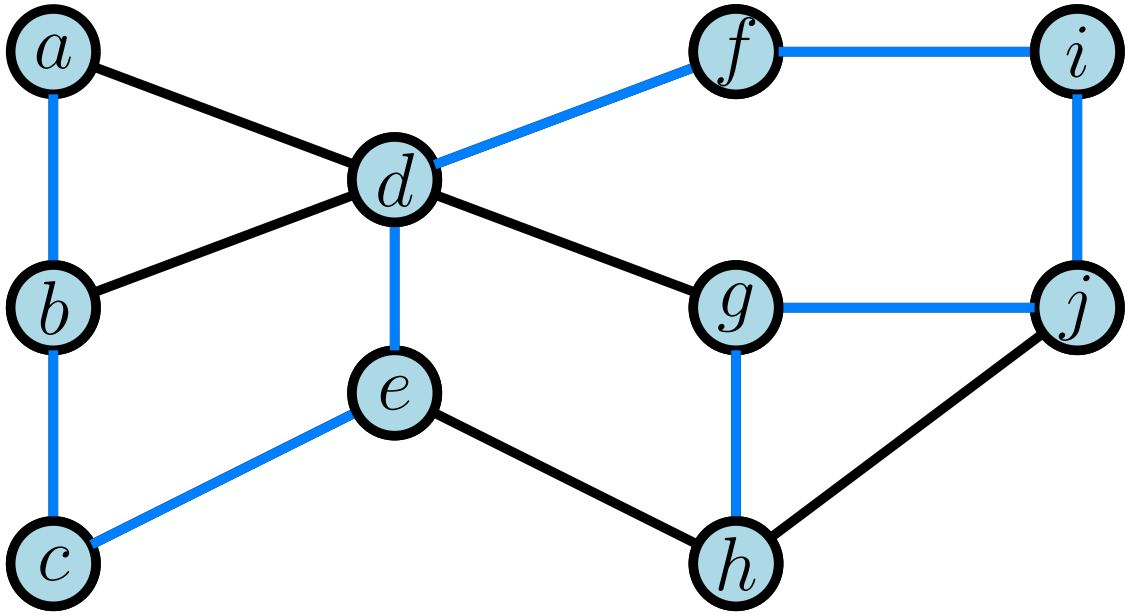
Sean $G = (V, E)$ un grafo simple y $r \in V$.

Regla de exploración:

Elegir $uv \in E$ con $u \in U$ y $v \notin U$, con u lo más nuevo posible.

- **Ej:**

$U = \{a, b, c, d, e, f, g, h, i, j\}$



$\tau(v)$	Arista	v	Nivel
0	—	a	0
1	ab	b	1
2	bc	c	2
3	ce	e	3
4	ed	d	4
5	df	f	5
6	fi	i	6
7	ij	j	7
8	jg	g	8
9	gh	h	9

- Implementación vía pila.

Árboles generadores de costo mínimo

Costos

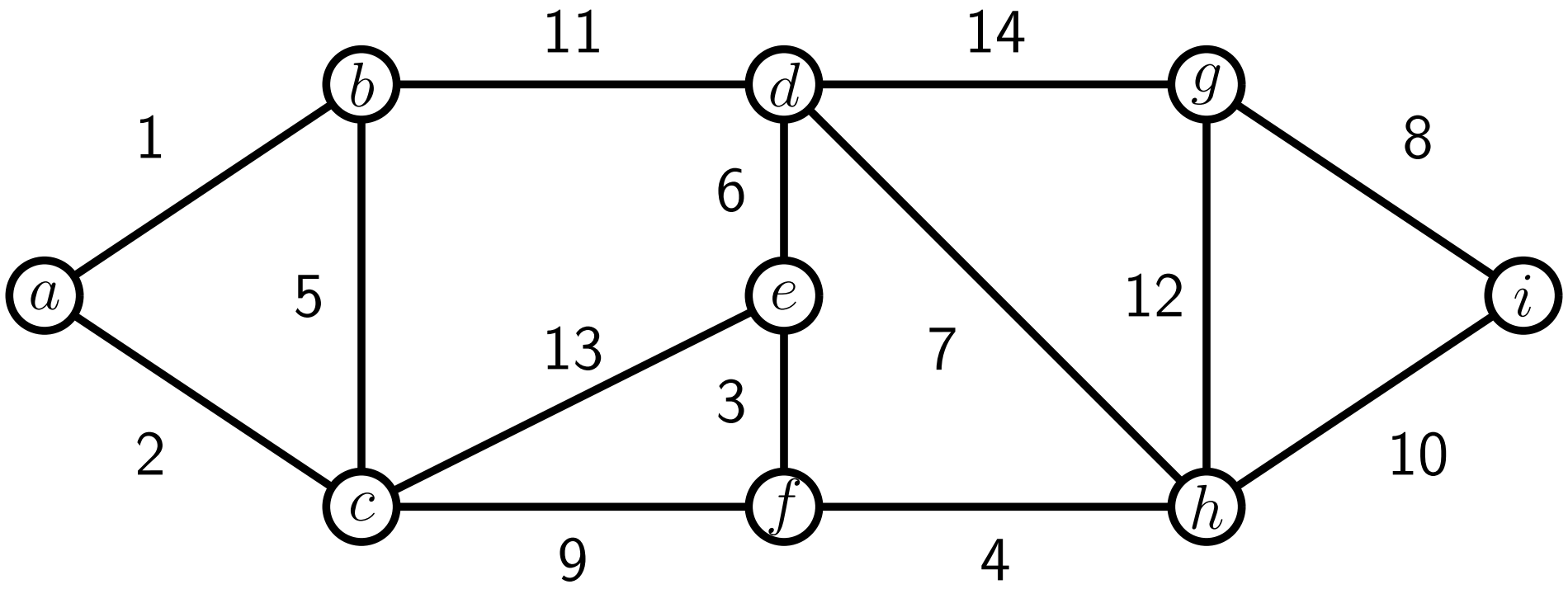
- Certificado de conexidad: aristas indistinguibles.

Costos

- Certificado de conexidad: aristas indistinguibles.
- Queremos certificar conexidad **al menor costo**

Costos

- Certificado de conexidad: aristas indistinguibles.
- Queremos certificar conexidad **al menor costo**
- ¿Aplicaciones/ejemplos?



Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

El *peso* de un subgrafo T es

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

El *peso* de un subgrafo T es

$$w(T) = \sum_{e \in E(T)} w(e).$$

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

El *peso* de un subgrafo T es

$$w(T) = \sum_{e \in E(T)} w(e).$$

Def.

Sea G un grafo simple y conexo con pesos w.

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

El *peso* de un subgrafo T es

$$w(T) = \sum_{e \in E(T)} w(e).$$

Def.

Sea G un grafo simple y conexo con pesos w.

Un *árbol generador de costo mínimo (MST)* es un árbol generador T de G que minimiza $w(T)$.

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

El *peso* de un subgrafo T es

$$w(T) = \sum_{e \in E(T)} w(e).$$

Def.

Sea G un grafo simple y conexo con pesos w.

Un *árbol generador de costo mínimo (MST)* es un árbol generador T de G que minimiza $w(T)$.

- Supondremos **pesos distintos**.

Árboles generadores de costo mínimo

Def.

Sea $G = (V, E)$ un grafo simple.

Un *grafo con pesos* es G junto con una función $w : E \to \mathbb{R}$.

El *peso* de un subgrafo T es

$$w(T) = \sum_{e \in E(T)} w(e).$$

Def.

Sea G un grafo simple y conexo con pesos w.

Un *árbol generador de costo mínimo (MST)* es un árbol generador T de G que minimiza $w(T)$.

- Supondremos **pesos distintos**.
  - Despues veremos porqué esto es SPG.

Unicidad

- **Consecuencias** de pesos distintos.

Unicidad

- **Consecuencias** de pesos distintos.

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Unicidad

- **Consecuencias** de pesos distintos.

Teo.

Sea G un grafo simple y conexo con pesos distintos.

G tiene un único árbol generador de costo mínimo.

Unicidad

- **Consecuencias** de pesos distintos.

Teo.

Sea G un grafo simple y conexo con pesos distintos.

G tiene un único árbol generador de costo mínimo.

- **Dem:**

Unicidad

- **Consecuencias** de pesos distintos.

Teo.

Sea G un grafo simple y conexo con pesos distintos.

G tiene un único árbol generador de costo mínimo.

- **Dem:**

- Podemos hablar de **el** MST.

Unicidad

- **Consecuencias** de pesos distintos.

Teo.

Sea G un grafo simple y conexo con pesos distintos.

G tiene un único árbol generador de costo mínimo.

- **Dem:**

- Podemos hablar de **el** MST.

- ¿Cómo encontrarlo?

Unicidad

- **Consecuencias** de pesos distintos.

Teo.

Sea G un grafo simple y conexo con pesos distintos.

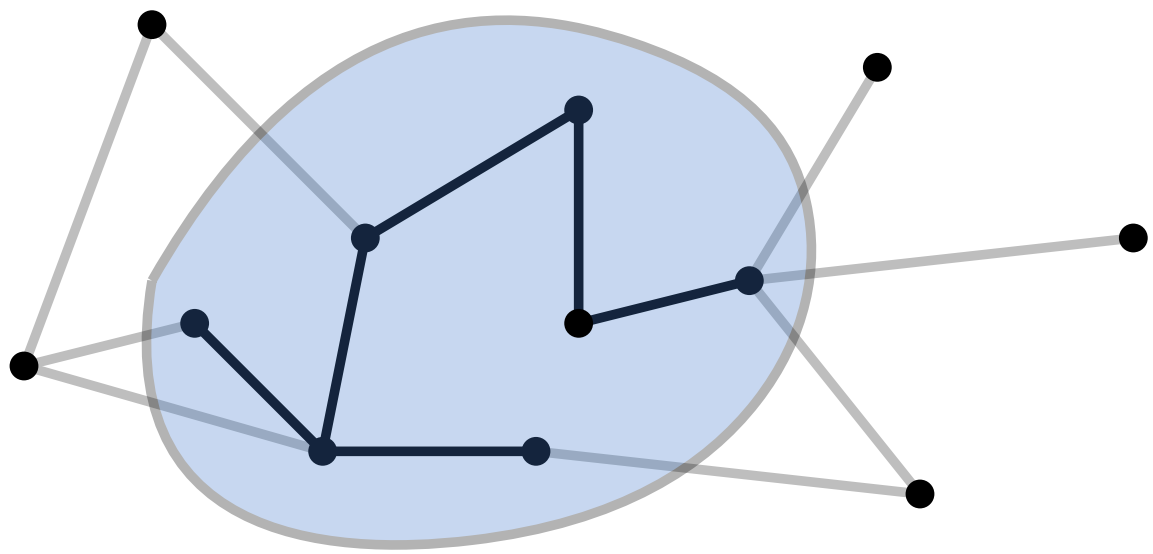
G tiene un único árbol generador de costo mínimo.

- **Dem:**

- Podemos hablar de **el** MST.
- ¿Cómo encontrarlo?
  - ¿Incremental?

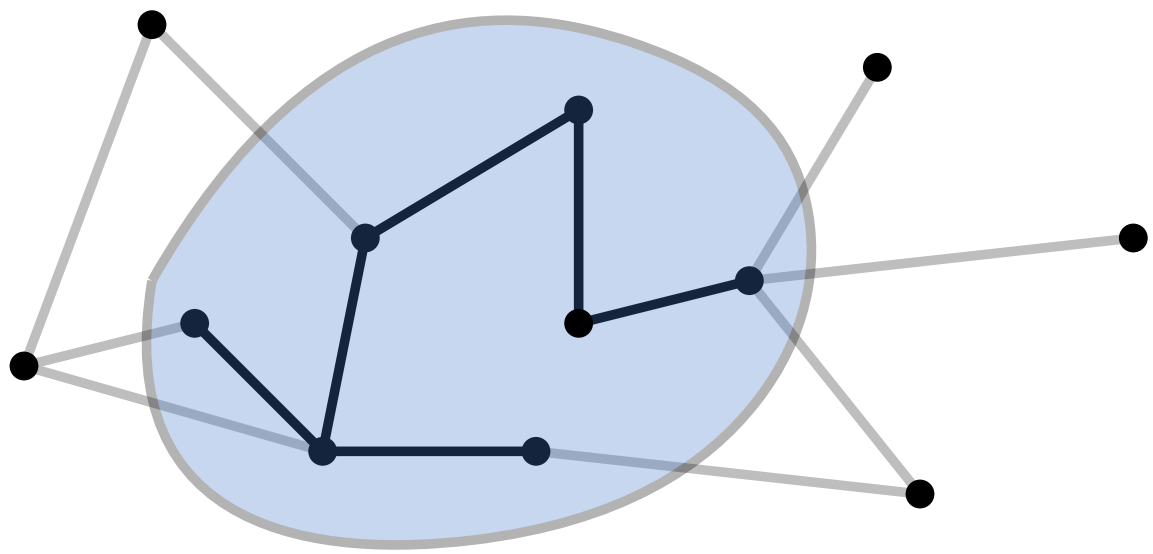
Regla azul

- ¿Cuál arista elegir?



Regla azul

- ¿Cuál arista elegir?

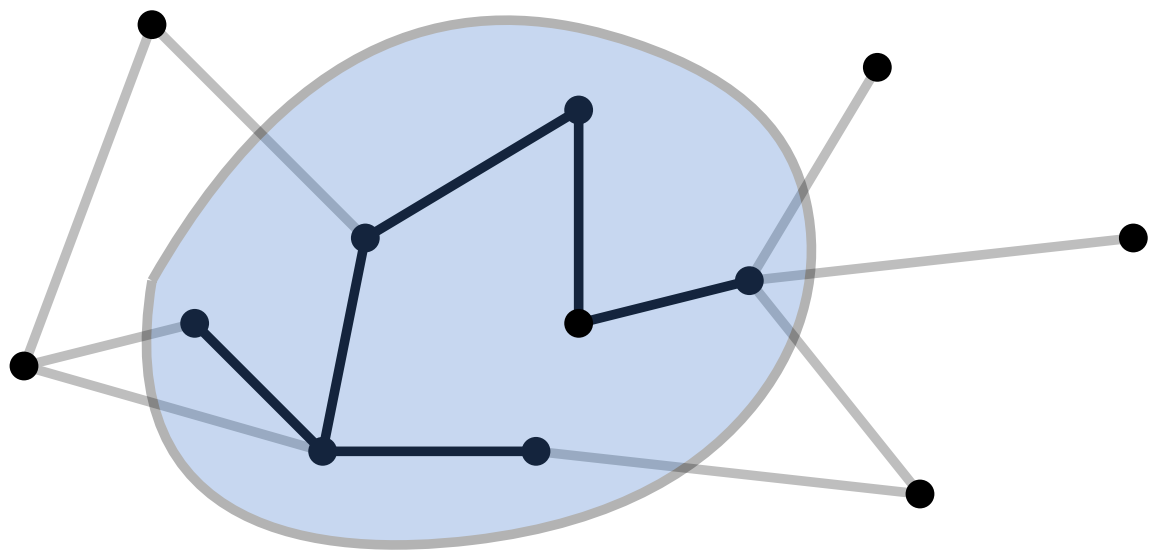


Regla azul

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos, y $U \subseteq V$.

Regla azul

- ¿Cuál arista elegir?



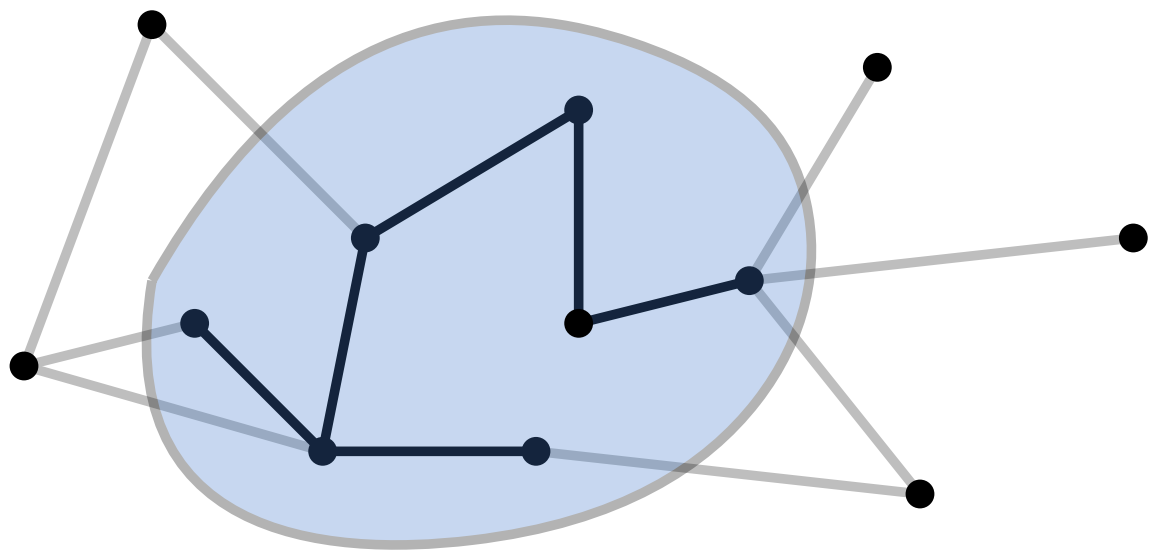
Regla azul

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos, y $U \subseteq V$.

Si $e = uv$ es la arista de menor peso con $u \in U$ y $v \notin U$, entonces e pertenece al MST.

Regla azul

- ¿Cuál arista elegir?



Regla azul

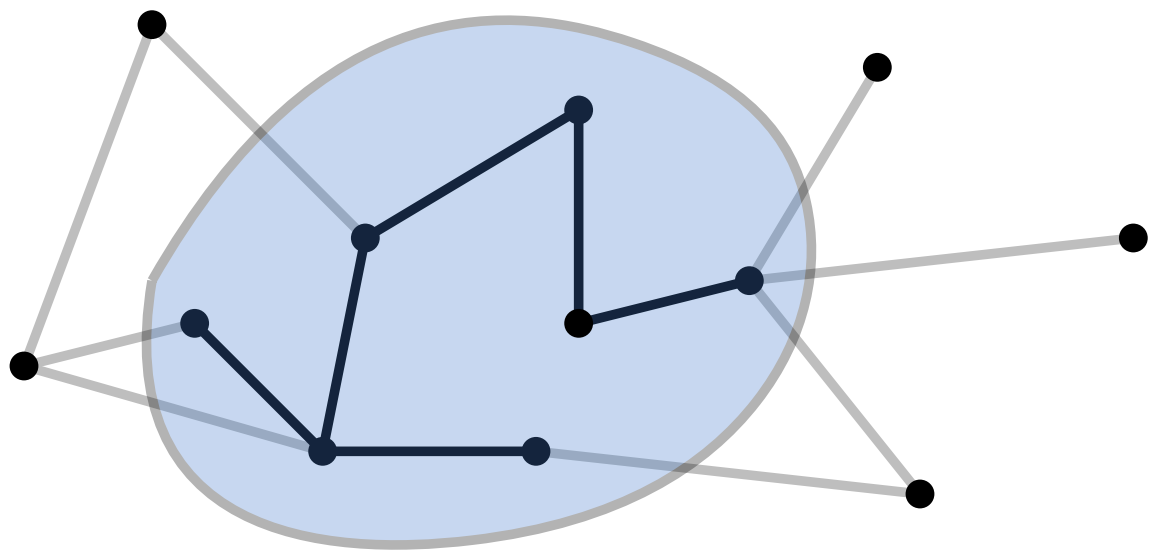
Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos, y $U \subseteq V$.

Si $e = uv$ es la arista de menor peso con $u \in U$ y $v \notin U$, entonces e pertenece al MST.

- **Dem:**

Regla azul

- ¿Cuál arista elegir?



Regla azul

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos, y $U \subseteq V$.

Si $e = uv$ es la arista de menor peso con $u \in U$ y $v \notin U$, entonces e pertenece al MST.

- **Dem:**
- ¡Sugiere un algoritmo!

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while $U \neq V$:

    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.

    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

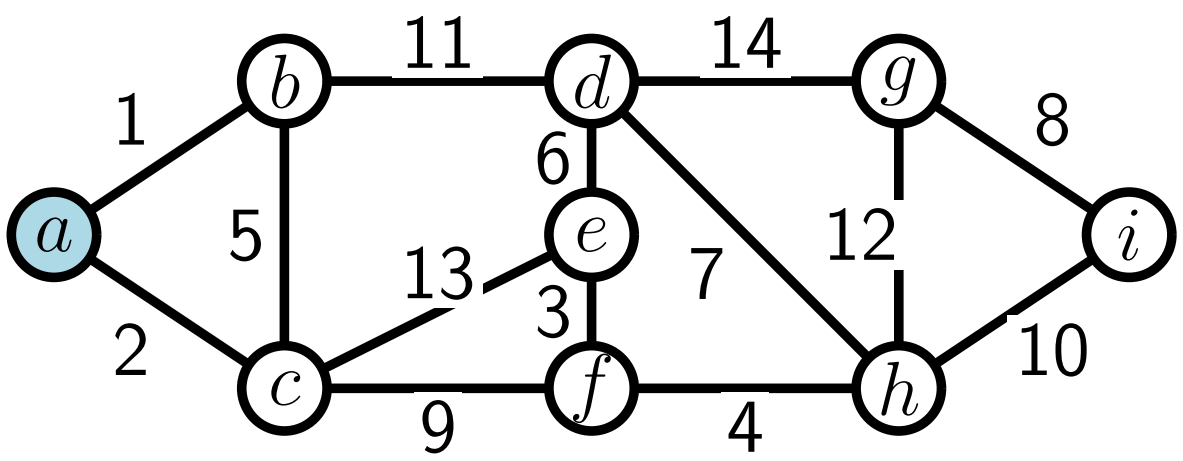
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a\}$$



Paso	Arista	$w(e)$
0	—	—

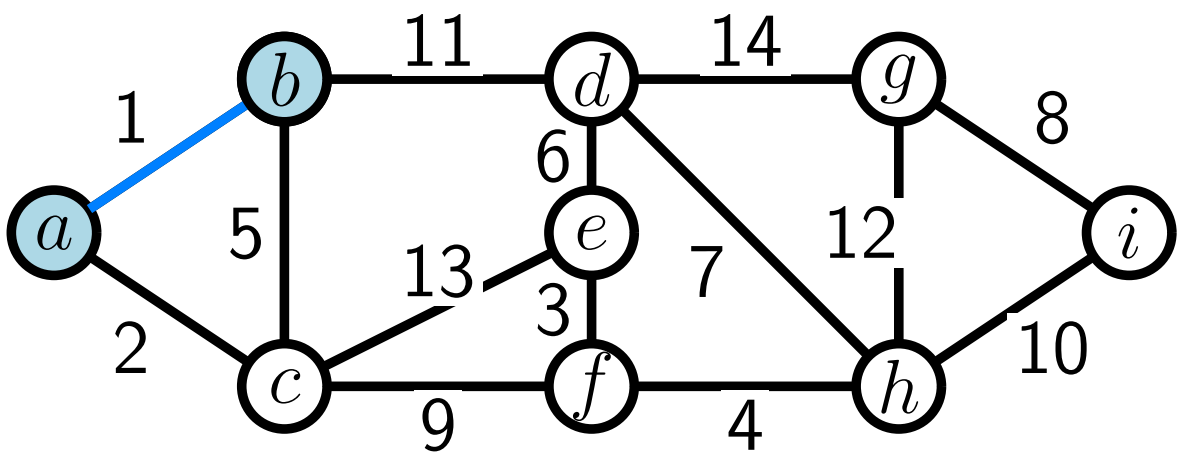
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1

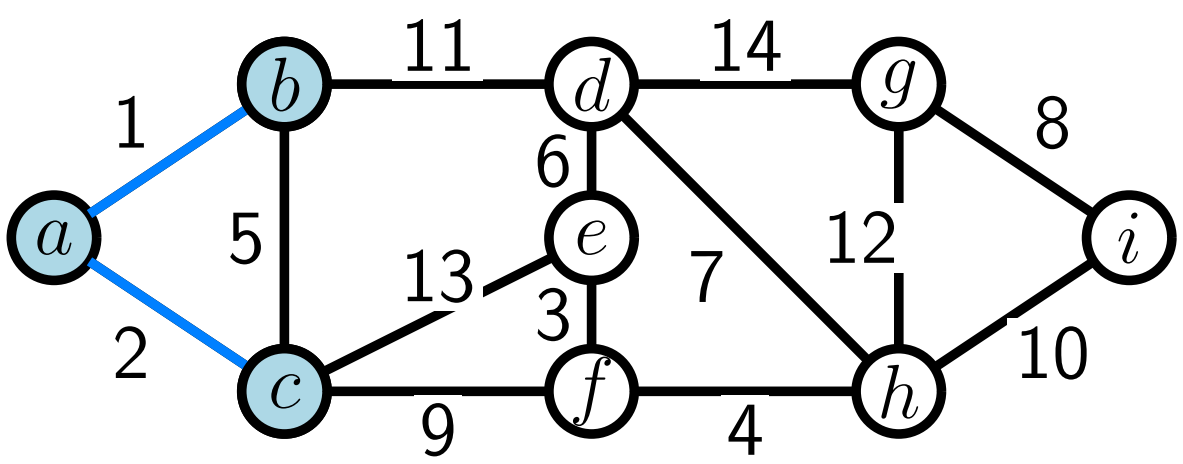
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$U = \{a, b, c\}$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2

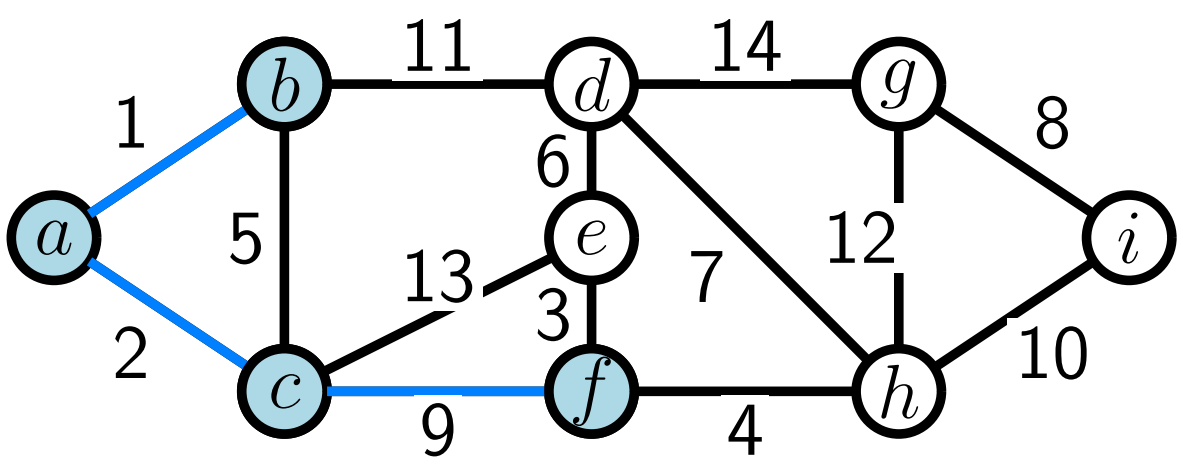
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b, c, f\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2
3	cf	9

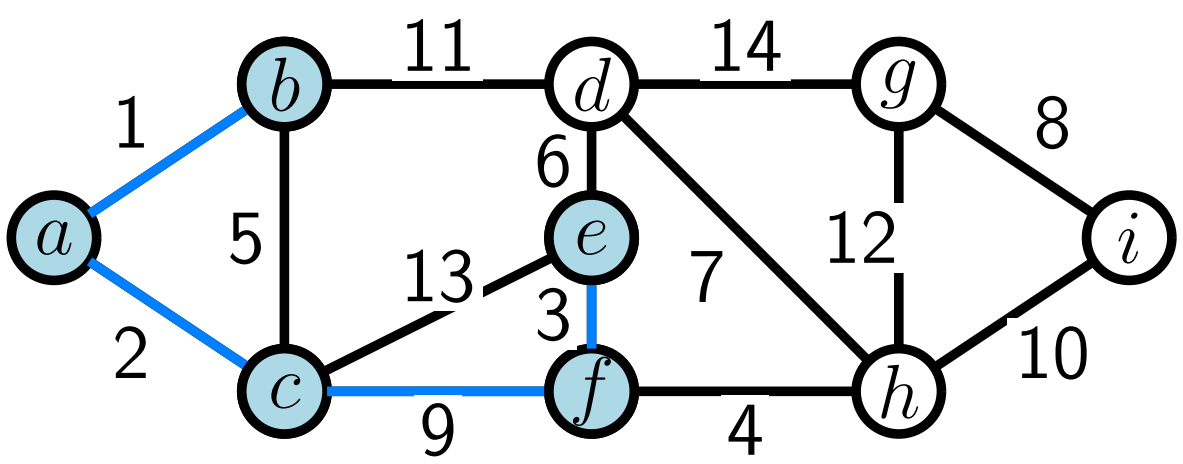
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b, c, e, f\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2
3	cf	9
4	ef	3

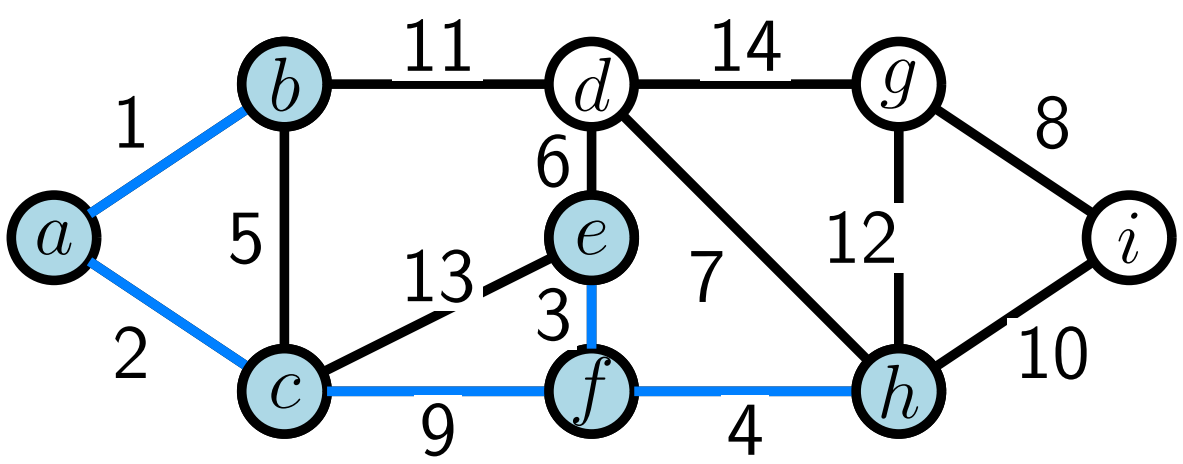
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b, c, e, f, h\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2
3	cf	9
4	ef	3
5	fh	4

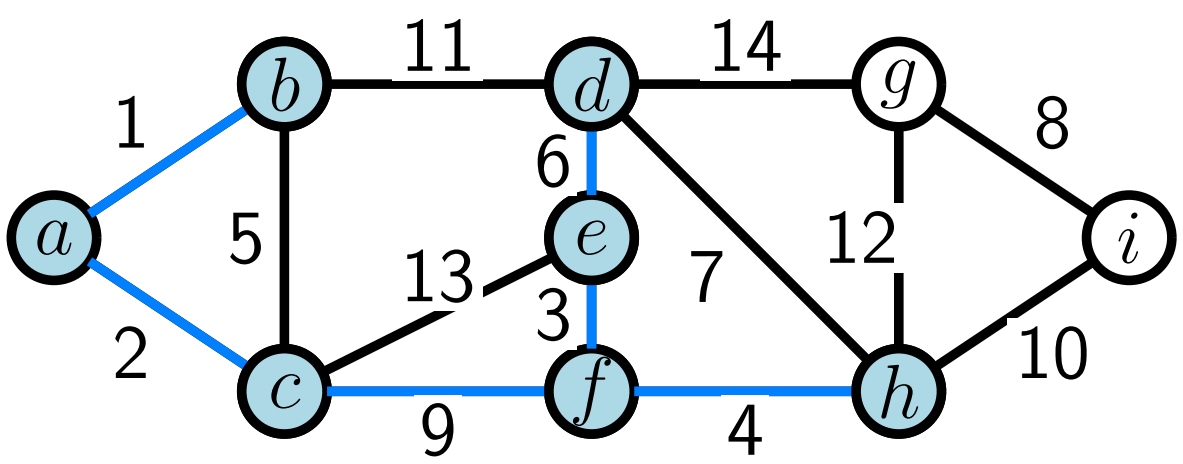
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b, c, d, e, f, h\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2
3	cf	9
4	ef	3
5	fh	4
6	de	6

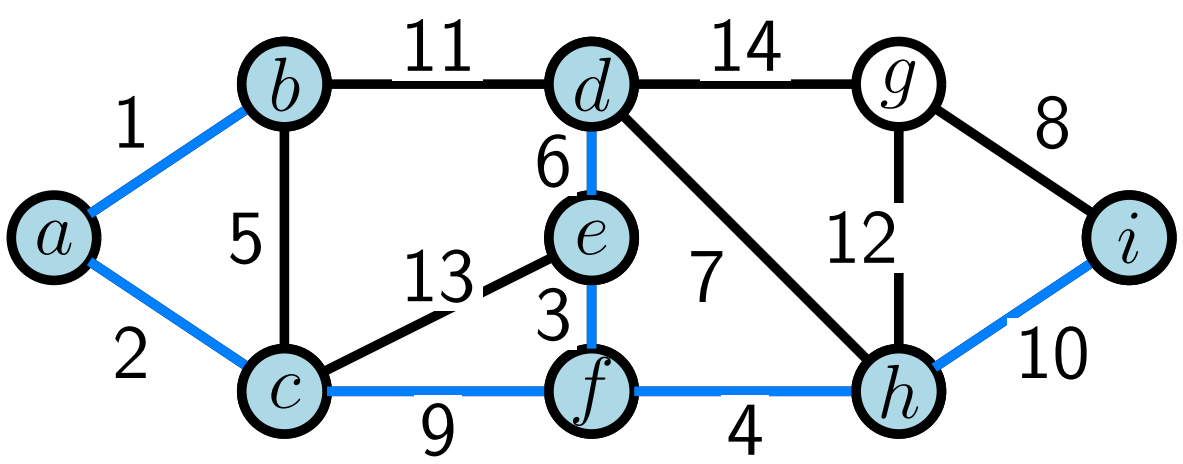
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b, c, d, e, f, h, i\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2
3	cf	9
4	ef	3
5	fh	4
6	de	6
7	hi	10

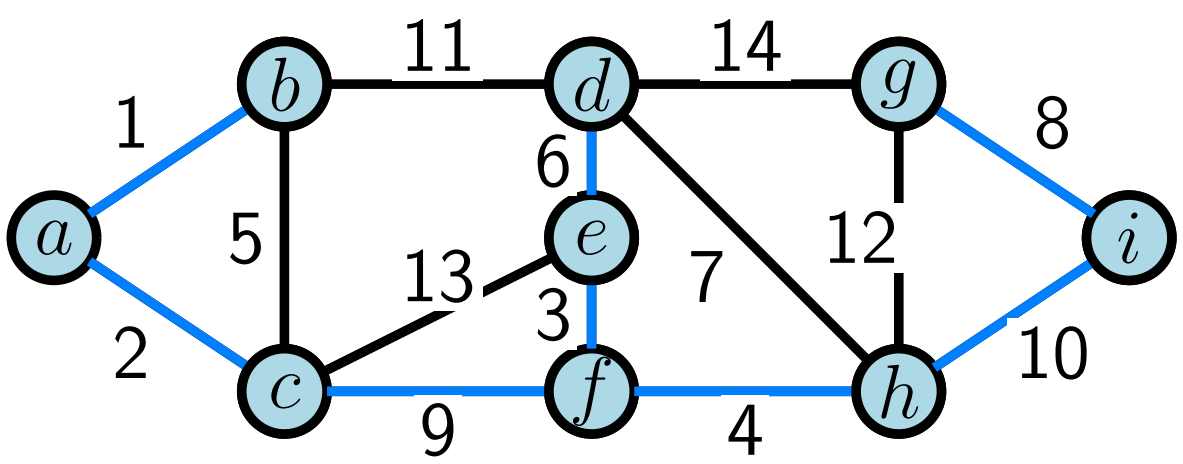
Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

- **Ej:**

$$U = \{a, b, c, d, e, f, g, h, i\}$$



Paso	Arista	$w(e)$
0	—	—
1	ab	1
2	ac	2
3	cf	9
4	ef	3
5	fh	4
6	de	6
7	hi	10
8	gi	8

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.
Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while $U \neq V$:

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while $U \neq V$:

Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while $U \neq V$:

Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.

$U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while $U \neq V$:

    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.

    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.

while $U \neq V$:

    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.

    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.

return (U, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Prim devuelve el árbol generador de costo mínimo de G.

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Prim devuelve el árbol generador de costo mínimo de G.

- **Dem:**

Algoritmo de Prim

Alg.

Sean $G = (V, E)$ un grafo simple y conexo con pesos distintos y $r \in V$.

Partir con $U \leftarrow \{r\}$ y $T \leftarrow \varnothing$.
while $U \neq V$:
    Elegir $uv \in E$ de menor peso con $u \in U$ y $v \notin U$.
    $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
return (U, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Prim devuelve el árbol generador de costo mínimo de G.

- **Dem:**
    - ¡Regla azul!

Regla roja

- La regla azul dice qué aristas debemos seleccionar.

Regla roja

- La regla azul dice qué aristas debemos seleccionar.
- ¿Hay algún motivo para evitar una arista?

Regla roja

- La regla azul dice qué aristas debemos seleccionar.
- ¿Hay algún motivo para evitar una arista?

Lem.

Sean G un grafo simple y conexo con pesos distintos y C el conjunto de aristas de un ciclo.

Regla roja

- La regla azul dice qué aristas debemos seleccionar.
- ¿Hay algún motivo para evitar una arista?

Lem.

Sean G un grafo simple y conexo con pesos distintos y C el conjunto de aristas de un ciclo.

La arista de mayor peso de C no pertenece al MST.

Regla roja

- La regla azul dice qué aristas debemos seleccionar.
- ¿Hay algún motivo para evitar una arista?

Lem.

Sean G un grafo simple y conexo con pesos distintos y C el conjunto de aristas de un ciclo.

La arista de mayor peso de C no pertenece al MST.

- **Dem:**

Regla roja

- La regla azul dice qué aristas debemos seleccionar.
- ¿Hay algún motivo para evitar una arista?

Lem.

Sean G un grafo simple y conexo con pesos distintos y C el conjunto de aristas de un ciclo.

La arista de mayor peso de C no pertenece al MST.

- **Dem:**
- ¡Sugiere un algoritmo!

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

        $T \leftarrow T \cup \{e\}$.

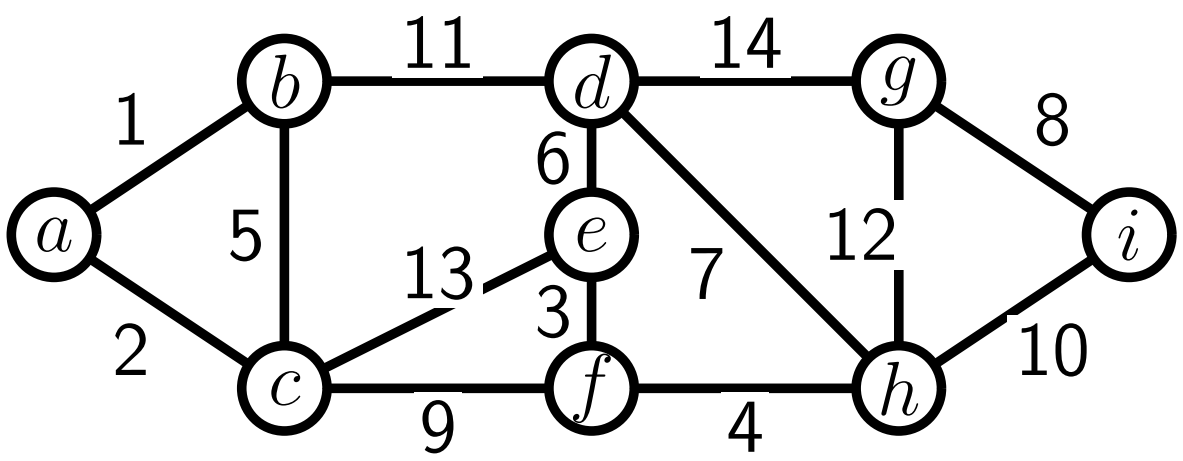
return (V, T).

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$

$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

  if $(V, T \cup \{e\})$ es acíclico:

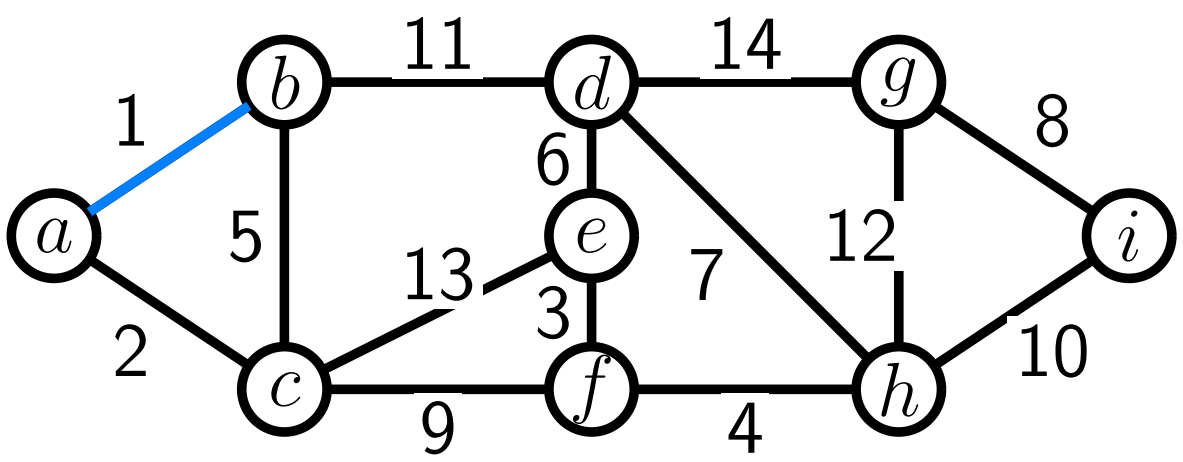
    $T \leftarrow T \cup \{e\}$.

return (V, T).

- **Ej:**

$w(e)$	e	$\in T$
1	ab	✓

$w(e)$	e	$\in T$

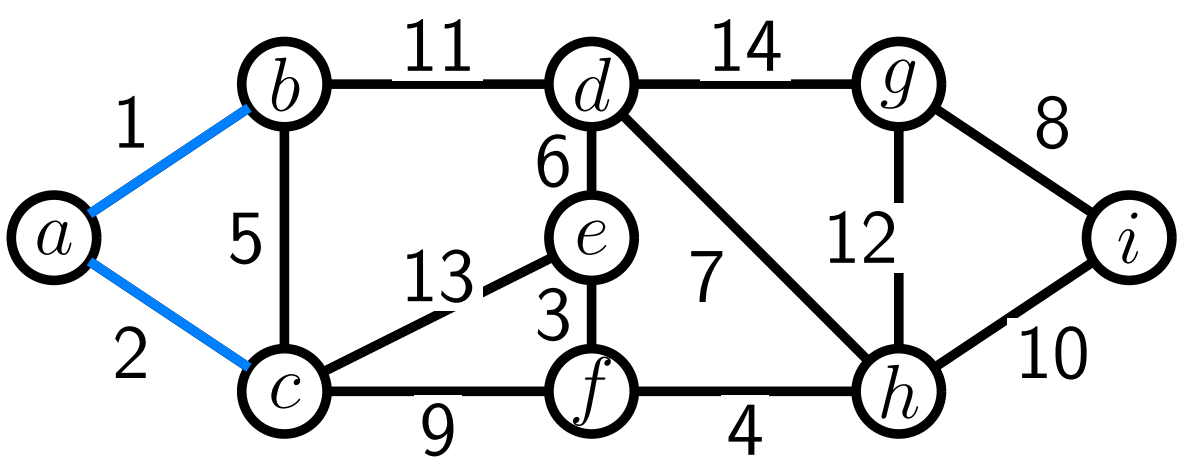


Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓

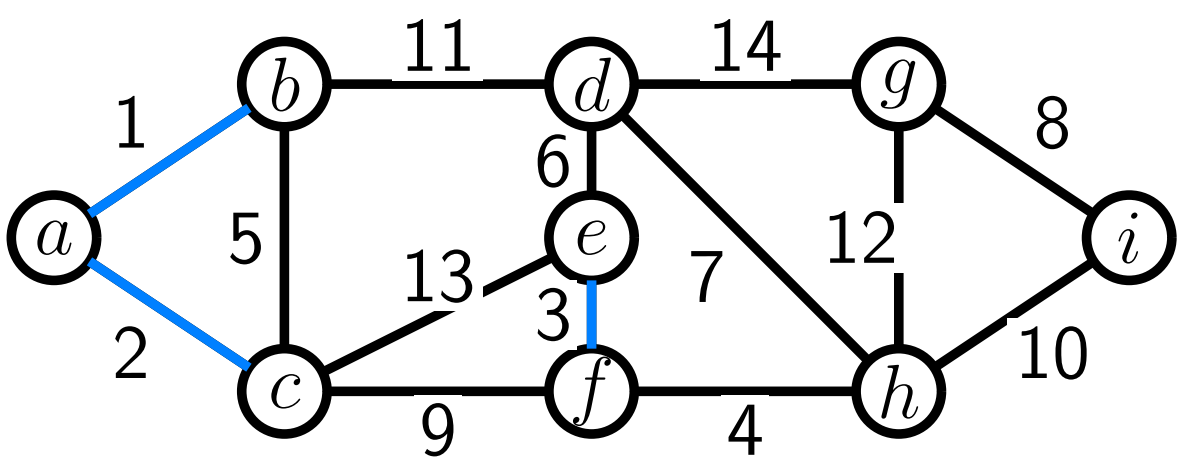
$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓

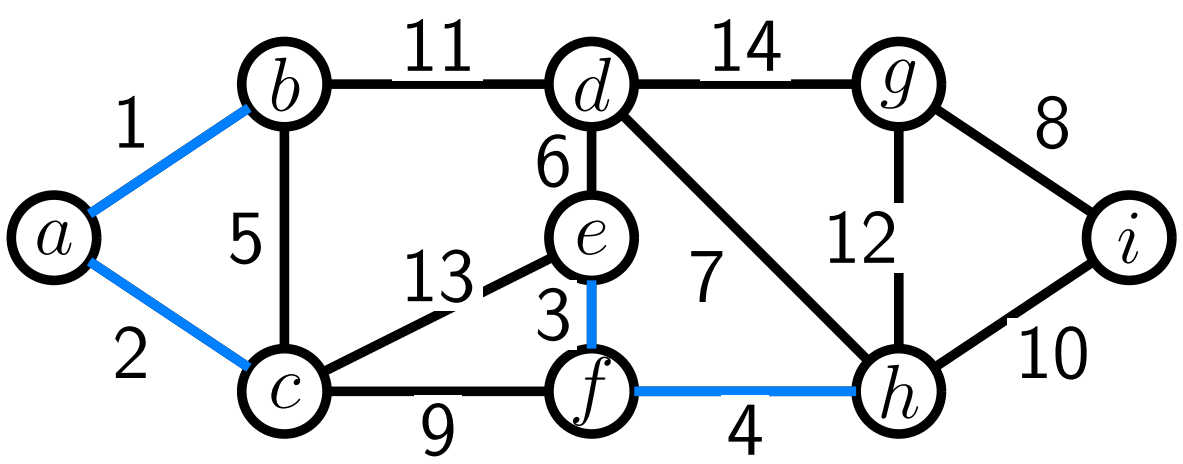
$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓

$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

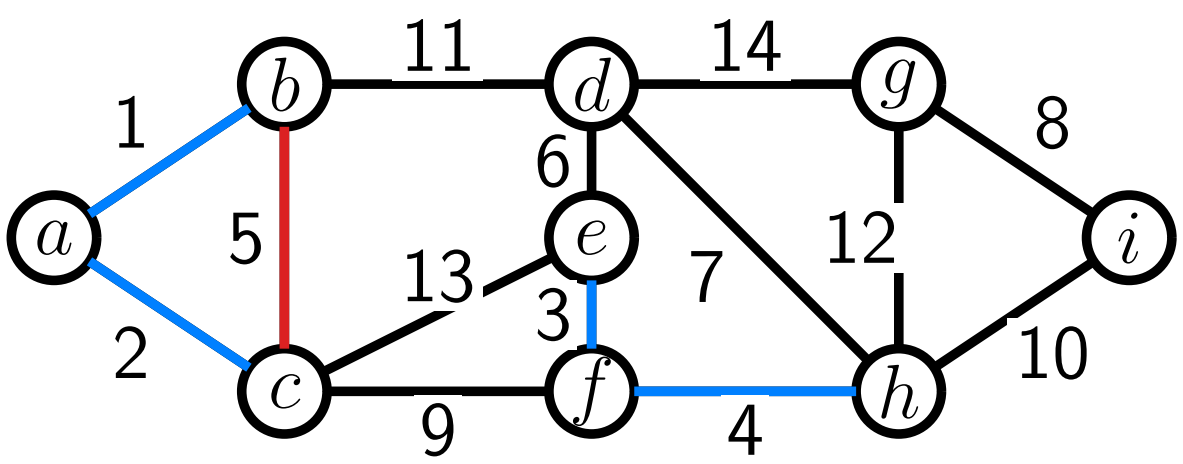
for $e \in E$ en ese orden:

  if $(V, T \cup \{e\})$ es acíclico:

    $T \leftarrow T \cup \{e\}$.

return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗

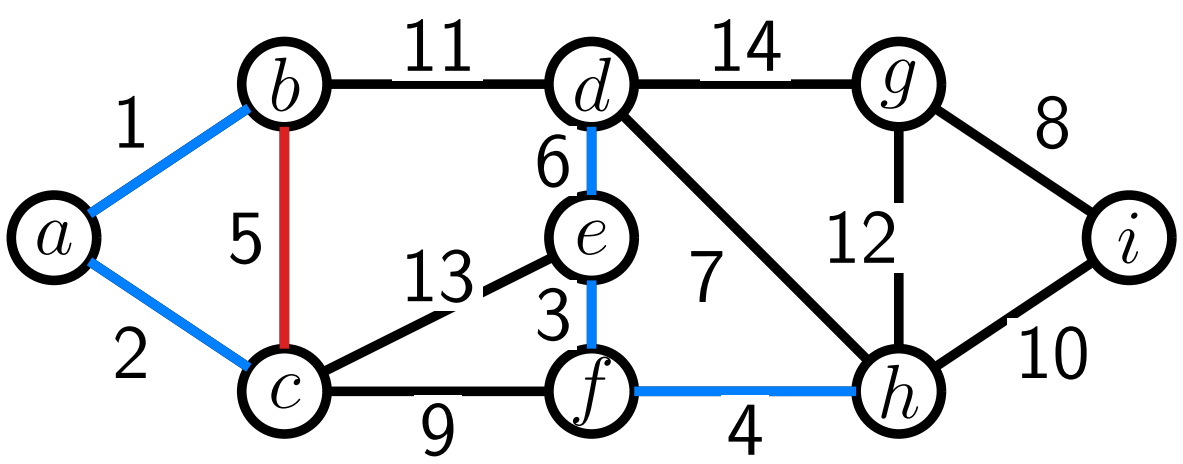
$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓

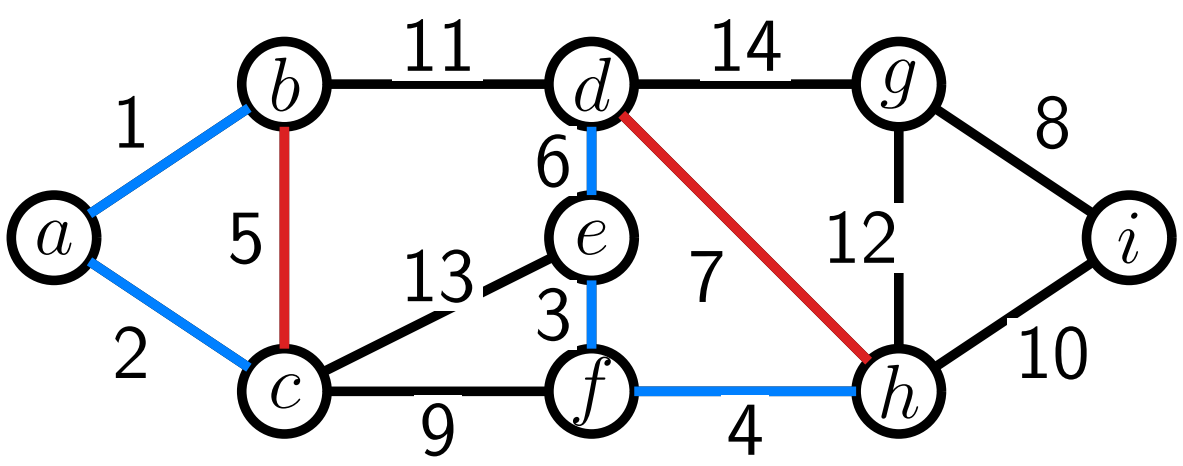
$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

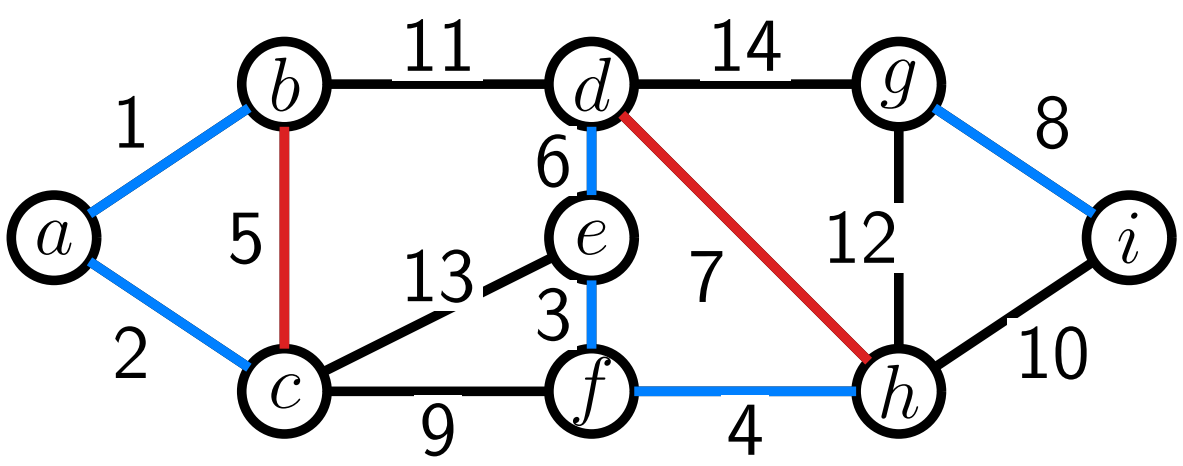
$w(e)$	e	$\in T$

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

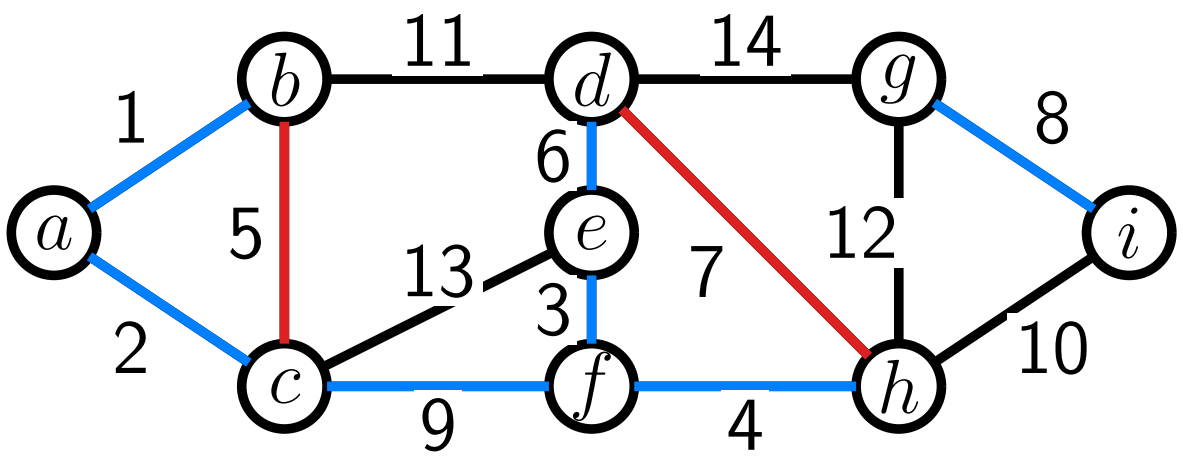
$w(e)$	e	$\in T$
8	gi	✓

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

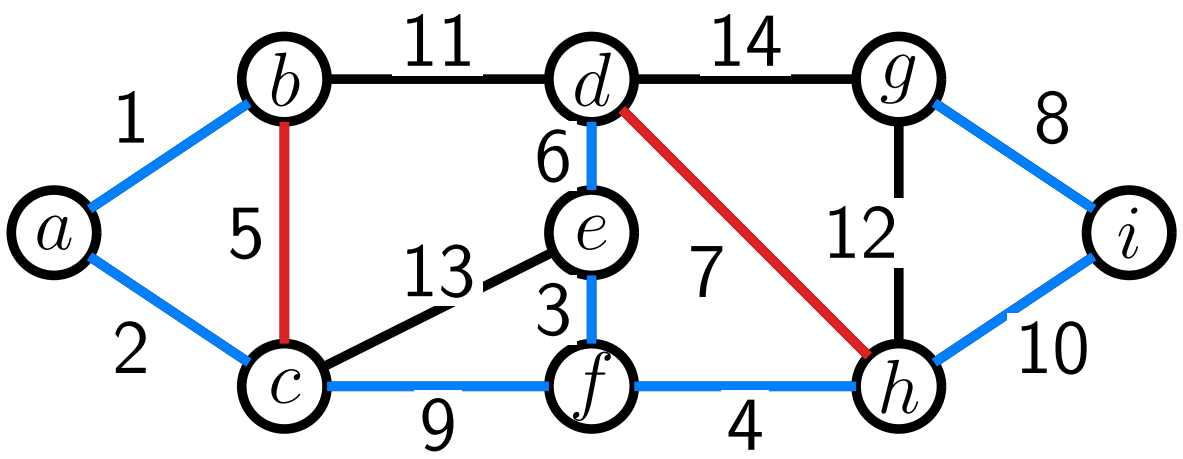
$w(e)$	e	$\in T$
8	gi	✓
9	cf	✓

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

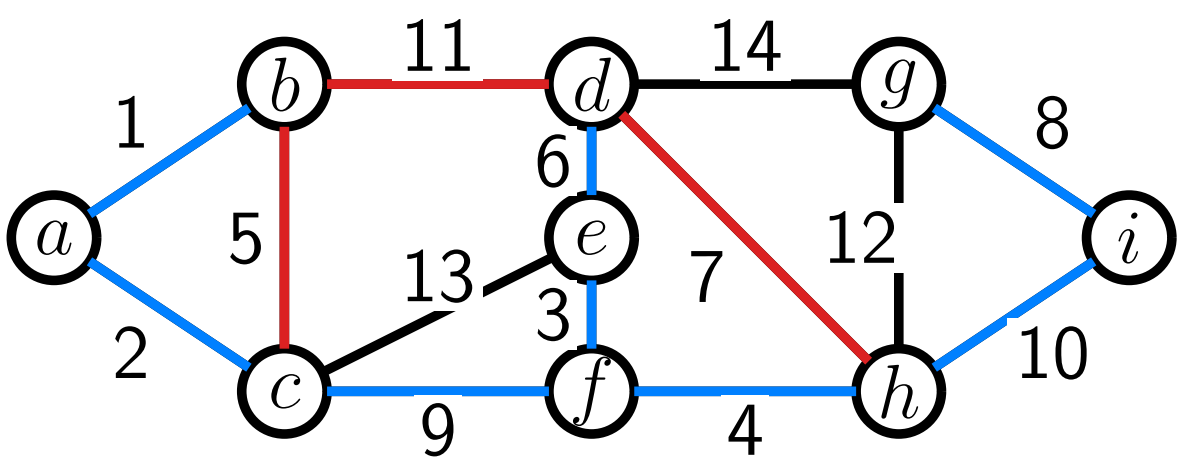
$w(e)$	e	$\in T$
8	gi	✓
9	cf	✓
10	hi	✓

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

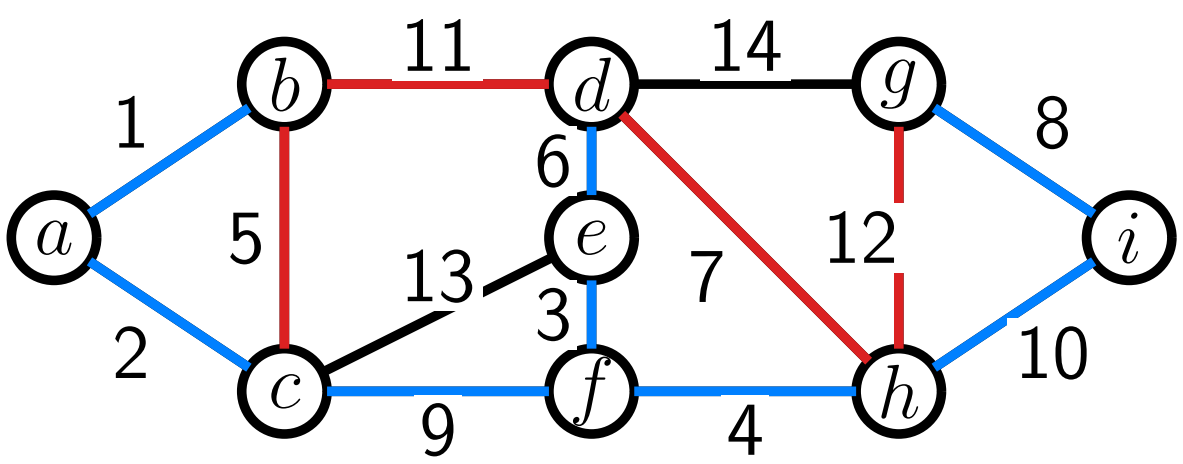
$w(e)$	e	$\in T$
8	gi	✓
9	cf	✓
10	hi	✓
11	bd	✗

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

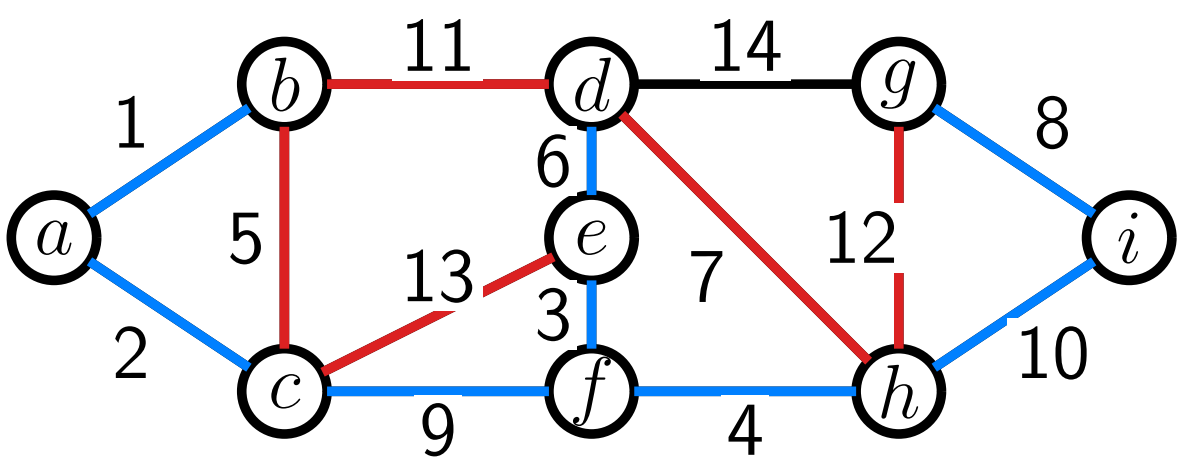
$w(e)$	e	$\in T$
8	gi	✓
9	cf	✓
10	hi	✓
11	bd	✗
12	gh	✗

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

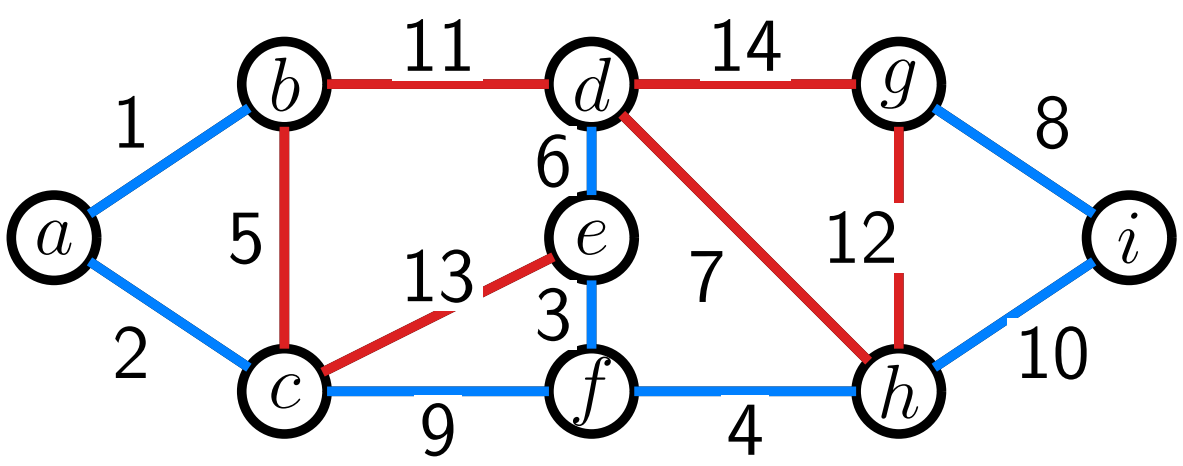
$w(e)$	e	$\in T$
8	gi	✓
9	cf	✓
10	hi	✓
11	bd	✗
12	gh	✗
13	ce	✗

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.
for $e \in E$ en ese orden:
    if $(V, T \cup \{e\})$ es acíclico:
        $T \leftarrow T \cup \{e\}$.
return (V, T).

- **Ej:**



$w(e)$	e	$\in T$
1	ab	✓
2	ac	✓
3	ef	✓
4	fh	✓
5	bc	✗
6	de	✓
7	dh	✗

$w(e)$	e	$\in T$
8	gi	✓
9	cf	✓
10	hi	✓
11	bd	✗
12	gh	✗
13	ce	✗
14	dg	✗

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.
Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

        $T \leftarrow T \cup \{e\}$.

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

        $T \leftarrow T \cup \{e\}$.

return (V, T).

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

        $T \leftarrow T \cup \{e\}$.

return (V, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

        $T \leftarrow T \cup \{e\}$.

return (V, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Kruskal devuelve el árbol generador de costo mínimo de G.

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

    if $(V, T \cup \{e\})$ es acíclico:

        $T \leftarrow T \cup \{e\}$.

return (V, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Kruskal devuelve el árbol generador de costo mínimo de G.

- **Dem:**

Algoritmo de Kruskal

Alg.

Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos.

Ordenar E por peso creciente; partir con $T \leftarrow \varnothing$.

for $e \in E$ en ese orden:

  if $(V, T \cup \{e\})$ es acíclico:

    $T \leftarrow T \cup \{e\}$.

return (V, T).

Teo.

Sea G un grafo simple y conexo con pesos distintos.

Kruskal devuelve el árbol generador de costo mínimo de G.

- **Dem:**
  - ¡Regla roja!

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?
- **Obs.:** Prim y Kruskal siguen funcionando al romper los empates **arbitrariamente**.

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?
- **Obs.:** Prim y Kruskal siguen funcionando al romper los empates **arbitrariamente**.
- Elejir $\varepsilon > 0$ pequeño y definir

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?
- **Obs.:** Prim y Kruskal siguen funcionando al romper los empates **arbitrariamente**.
- Elejir $\varepsilon > 0$ pequeño y definir

$$w_\varepsilon(e_i) = w(e_i) + i\varepsilon.$$

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?
- **Obs.:** Prim y Kruskal siguen funcionando al romper los empates **arbitrariamente**.
- Elejir $\varepsilon > 0$ pequeño y definir

$$w_\varepsilon(e_i) = w(e_i) + i\varepsilon.$$

- El MST para w_ε es un MST para w.

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?
- **Obs.:** Prim y Kruskal siguen funcionando al romper los empates **arbitrariamente**.
- Elejir $\varepsilon > 0$ pequeño y definir

$$w_\varepsilon(e_i) = w(e_i) + i\varepsilon.$$

- El MST para w_ε es un MST para w.
- Las reglas también dan otros algoritmos:

Empates y otros algoritmos

- ¿Qué pasa con pesos no distintos?
- **Obs.:** Prim y Kruskal siguen funcionando al romper los empates **arbitrariamente**.
- Elejir $\varepsilon > 0$ pequeño y definir

$$w_\varepsilon(e_i) = w(e_i) + i\varepsilon.$$

- El MST para w_ε es un MST para w.
- Las reglas también dan otros algoritmos:
  - Round Robin, Borůvka, Kruskal inverso.

Dudas/Ejercicios