# 4.4. Árboles generadores

En la sección anterior vimos que un árbol es el certificado, en cierto sentido, *óptimo* de la conexidad. Otra forma alternativa de verlo, es que si yo quiero certificar que *recorrí* todos los vértices accesibles de un grafo, lo mínimo que puedo hacer es exhibir un árbol que los conecte a todos. Una tercera forma es pensar que la forma más económica de conectar todos los vértices de un grafo es vía un árbol.

En esta sección mostraremos varias cosas relacionadas con esta idea, desde una perspectiva discreta y algorítmica. (1) si nos dan un grafo, como encontrar un árbol que certifique su conexidad, (2) de manera equivalente, una forma de explorar las componentes conexas de un grafo, y (3) si además nos dan costos asociados a las aristas, cómo encontrar una forma de conectar todos los vértices de la manera más barata posible.

La perspectiva de la sección mostrará una interacción interesante: desarrollaremos primero ideas estructurales discretas, y luego veremos como naturalmente dichas ideas se traducen en algoritmos.

## 4.4.1. Árboles generadores y algoritmos de exploración

Partimos introduciendo la noción del árbol generador, que es el certificado de conexidad que nos interesa.

> **Definición.** Sea $G$ un grafo simple. Un *árbol generador* de $G$ es un subgrafo $T$ de $G$ tal que:
>
> - $T$ contiene todos los vértices de $G$;
> - $T$ es un árbol.

La palabra "generador" indica que no estamos descartando vértices: el subgrafo debe usar todo el conjunto de vértices de $G$.

Si $G$ tiene $n$ vértices, entonces cualquier árbol generador de $G$ tiene exactamente $n-1$ aristas.

**Ejemplo.** El ciclo $C_n$ tiene $n$ árboles generadores distintos. En efecto, al borrar una de sus $n$ aristas obtenemos uno de ellos. Lo que queda es un camino con $n$ vértices, que es un árbol generador de $C_n$. Recíprocamente, un árbol generador debe conservar $n-1$ de las $n$ aristas del ciclo, así que todos se obtienen de esta forma.

En cambio, si un grafo no es conexo, no puede tener árbol generador. Ningún subgrafo puede crear una conexión entre vértices que ya estaban en componentes distintas del grafo original.

> **Teorema.** Sea $G$ un grafo simple. Entonces $G$ tiene un árbol generador si y sólo si $G$ es conexo.

*Demostración.* Supongamos primero que $G$ tiene un árbol generador $T$. Como $T$ es un árbol, entre cada par de vértices existe un camino en $T$. Como $T$ es un subgrafo de $G$, ese mismo camino también existe en $G$. Por lo tanto, $G$ es conexo.

Para la otra dirección, consideramos $H = (V, E')$ un subgrafo de $G$ sea conexo y con una cantidad maximal de aristas. Si $H$ no es un árbol, entonces contiene un ciclo. Luego, podemos eliminar una de sus aristas y el subgrafo sigue siendo conexo, contradiciendo la maximalidad de $H$. Por tanto, $H$ es un árbol con todos los vértices de $G$, es decir, un árbol generador. □

Notemos que en la demostración anterior *construimos* un árbol generador borrando las aristas redundantes.

Para diseñar algoritmos, sin embargo, suele ser más natural construir algoritmos de manera *iterativa*; es decir, trataremos de mantener un árbol invariante a lo largo de la ejecución del algoritmo, y lo haremos crecer paso a paso hasta que se convierta en un árbol generador.

### 4.4.2. Exploración incremental

¿Cómo podemos hacer crecer un árbol paso a paso? Una forma natural es agregar un nuevo vértice con exactamente una arista que lo conecte a un vértice ya incorporado: dicha construcción no genera ciclos y mantiene la conectividad.

> **Lema.** Sea $G = (V, E)$ un grafo simple, $U \subseteq V$ y $(U, T)$ subgrafo árbol de $G$. Si $e = uv$ conecta $u \in U$ y $v \notin U$, entonces $(U \cup \{v\}, T \cup \{uv\})$ es un árbol.

*Demostración.* Sea $T' = (U \cup \{v\}, T \cup \{uv\})$. La conectividad se conserva por la forma en que construimos $T'$. Los vértices de $U$ ya estaban conectados entre sí en $(U, T)$, y el nuevo vértice $v$ queda conectado a ellos mediante la arista $uv$. Por lo tanto, todos los vértices de $T'$ quedan en una misma componente conexa.

Falta ver que no aparece un ciclo. El grafo $(U, T)$ no tenía ciclos. Cualquier ciclo nuevo tendría que usar la arista $uv$, porque es la única arista que agregamos. Para cerrar un ciclo con $uv$, tendría que existir además un camino desde $v$ hasta $u$ usando sólo aristas de $(U, T)$. Eso es imposible, pues $v \notin U$ y $(U, T)$ sólo contiene vértices de $U$.

Así, $T'$ es conexo y acíclico. Por lo tanto, es un árbol. □

Lo anterior sugiere un algoritmo de manera natural: iterativamente agregar vértices via el lema anterior hasta que no sea posible.

**Algoritmo 1** Exploración

```
1: Entrada: un grafo simple G = (V, E) y una raíz r ∈ V.
2: Salida: un árbol generador de la componente conexa de r.
3: Partir con U ← {r} y T ← ∅.
4: while existe uv ∈ E con u ∈ U y v ∉ U do
5:     Elegir uv ∈ E con u ∈ U y v ∉ U.
6:     U ← U ∪ {v}, T ← T ∪ {uv}.
7: end while
8: return (U, T).
```

La gracia del algoritmo es que siempre funciona, sin importar cómo se elija la arista entre $U$ y $U^c$. Para identificar qué hemos recorrido al detenernos, observemos qué ocurre cuando un subgrafo conexo ya no tiene aristas hacia el resto del grafo.

> **Lema.** Sea $G = (V, E)$ un grafo simple y sea $H = (U, F)$ un subgrafo conexo de $G$. Si no existe una arista $uv \in E$ con $u \in U$ y $v \notin U$, entonces $U$ es el conjunto de vértices de una componente conexa de $G$.

*Demostración.* Como $H$ es conexo, está contenido en una componente conexa $C$ de $G$. Si $C$ tuviera un vértice fuera de $U$, un camino desde un vértice de $U$ hasta él tendría una primera arista que sale de $U$, contradiciendo la hipótesis. Luego $C$ tiene exactamente los vértices de $U$. □

El paso incremental mantiene conectados los vértices que vamos visitando. El lema anterior nos permitirá reconocerlos como el conjunto de vértices de una componente cuando la exploración termine.

> **Teorema.** Sea $G = (V, E)$ un grafo simple y sea $r \in V$. El algoritmo de exploración genérico devuelve un árbol generador de la componente conexa de $r$. En particular, si $G$ es conexo, devuelve un árbol generador de $G$.

*Demostración.* La demostración es por invariante. Después de cada paso, el subgrafo $(U, T)$ es un árbol.

Al comienzo, $U = \{r\}$ y $T = \varnothing$. Por lo tanto, $(U, T)$ es el árbol con un único vértice. Si en algún paso agregamos una arista $uv$ con $u \in U$ y $v \notin U$, entonces por el lema del paso incremental el nuevo subgrafo sigue siendo un árbol.

En cada paso agregamos un vértice nuevo a $U$. Como $V$ es finito, el proceso termina. En ese momento, no existe una arista con un extremo en $U$ y otro fuera de $U$, por la condición de término. Como $(U, T)$ es conexo, el lema anterior nos dice que $U$ es el conjunto de vértices de una componente conexa de $G$. Como $r \in U$, esa componente es la de $r$, y $(U, T)$ es un árbol generador de ella. Si $G$ es conexo, esa componente es todo $G$. □

### 4.4.3. Búsqueda en anchura y búsqueda en profundidad

El algoritmo de búsqueda genérico nos dice que funciona cualquier regla de elección de aristas. Hay dos reglas naturales que producen algoritmos clásicos: la búsqueda en anchura (BFS) y la búsqueda en profundidad (DFS).

Ambas reglas son elegidas según qué vértice de $U$ se expande en cada paso: una usa el vértice que llegó primero, y la otra el que llegó último.

> **Definición** (Tiempo de llegada)**.** En una ejecución del algoritmo de exploración, el *tiempo de llegada* de un vértice $x$ es el instante en que $x$ se agrega al conjunto $U$. Denotamos este tiempo por $\tau(x)$. La raíz tiene tiempo de llegada 0 y, cada vez que se agrega un vértice nuevo, se le asigna el siguiente tiempo de llegada.

Primero consideremos la regla que siempre continúa desde el vértice de menor tiempo de llegada que todavía permite agregar un vértice nuevo.

**Definición** (Búsqueda en anchura (BFS))**.** La *búsqueda en anchura*, o *BFS*, es el algoritmo de exploración con la siguiente regla de elección. En cada paso escogemos una arista $uv \in E$ con $u \in U$ y $v \notin U$ tal que $u$ tenga el menor tiempo de llegada posible:

$$\tau(u) = \text{mín}\{\tau(x) : x \in U \text{ y existe } y \notin U \text{ tal que } xy \in E\}.$$

Luego agregamos la arista $uv$ y asignamos a $v$ el siguiente tiempo de llegada. Si hay varias aristas que cumplen la regla, cualquiera de ellas puede elegirse.

BFS explora completamente los vecinos de cada vértice antes de pasar al siguiente. Por eso su implementación natural usa una cola: los vértices que llegan primero son también los primeros que se expanden.

La búsqueda en anchura viene con una gracia adicional: nos permite medir distancias mínimas en el grafo.

**Definición.** Sea $G = (V, E)$ un grafo simple y sean $u, v \in V$. La *distancia* entre $u$ y $v$, denotada por $d_G(u, v)$, es el largo mínimo de un camino desde $u$ hasta $v$ en $G$, medido por su número de aristas.

El árbol de búsqueda sólo conserva algunas aristas de $G$. Aun así, veremos que el camino desde la raíz hasta cada vértice tiene el menor largo posible en el grafo original.

**Proposición** (BFS computa distancias mínimas)**.** Sea $G$ un grafo simple conexo y sea $r \in V(G)$. El nivel de $v$ en el árbol de búsqueda es exactamente $d_G(r, v)$.

*Demostración.* Sea $\ell(v)$ el nivel en que BFS incorpora a $v$. Primero justifiquemos que los niveles se recorren en orden. BFS expande los vértices en orden de llegada. Al expandir uno de nivel $j$, sus hijos nuevos tienen nivel $j + 1$ y quedan al final de la cola, detrás de los vértices de nivel $j$ que aún esperan. Partiendo de la raíz, por inducción se incorporan todos los vértices de nivel $j + 1$ antes de empezar a expandirlos y, por tanto, antes de incorporar alguno de nivel $j + 2$.

El árbol de BFS contiene un camino desde $r$ hasta $v$ de largo $\ell(v)$. Por lo tanto,

$$d_G(r, v) \leq \ell(v).$$

Probemos ahora la desigualdad contraria. Supongamos, buscando una contradicción, que existe un vértice $v$ tal que

$$d_G(r, v) < \ell(v).$$

Tomemos un contraejemplo de distancia mínima, y sea

$$r = x_0, x_1, \ldots, x_k = v$$

un camino más corto desde $r$ hasta $v$. Entonces $k = d_G(r, v)$.

El vértice $x_{k-1}$ tiene distancia $k-1$, así que por la minimalidad de $v$ no es un contraejemplo. Luego BFS lo incorpora en el nivel $k-1$. Cuando BFS explora los vértices de ese nivel, ve la arista $x_{k-1}v$. Por lo tanto, $v$ debió ser incorporado a nivel a lo más $k$. Es decir,

$$\ell(v) \leq k = d_G(r, v),$$

contradiciendo la elección de $v$.

Concluimos que $\ell(v) = d_G(r, v)$ para todo vértice $v$. □

La segunda regla natural siempre continúa desde el vértice de mayor tiempo de llegada que todavía permite agregar un vértice nuevo.

**Definición** (Búsqueda en profundidad (DFS))**.** La *búsqueda en profundidad*, o *DFS*, es el algoritmo de exploración con la siguiente regla de elección. En cada paso escogemos una arista $uv \in E$ con $u \in U$ y $v \notin U$ tal que $u$ tenga el mayor tiempo de llegada posible:

$$\tau(u) = \text{máx}\{\tau(x) : x \in U \text{ y existe } y \notin U \text{ tal que } xy \in E\}.$$

Luego agregamos la arista $uv$ y asignamos a $v$ el siguiente tiempo de llegada. Si hay varias aristas que cumplen la regla, cualquiera de ellas puede elegirse.

DFS explora lo más profundo posible antes de retroceder. Su implementación natural usa una pila: el último vértice que llega es el primero que se intenta expandir.

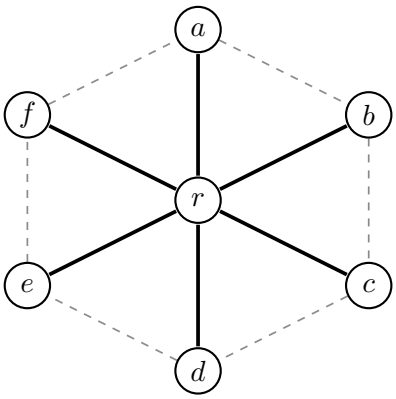
Cuando el vértice actual ya no tiene vecinos por visitar, retrocedemos por la rama recorrida hasta encontrar uno que permita continuar. Así, la búsqueda va completando una rama antes de comenzar la siguiente.

**Ejemplo.** El *grafo rueda* $W_n$ con $n \geq 4$ se obtiene tomando un ciclo con $n-1$ vértices y agregando un vértice central adyacente a todos los vértices del ciclo.

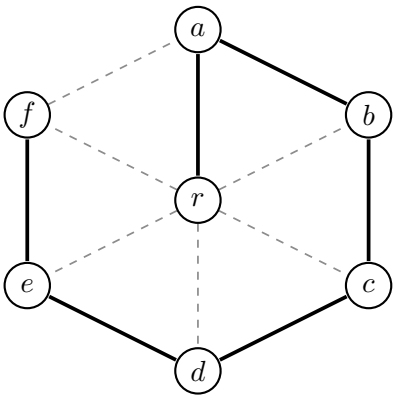
En una rueda $W_n$, si comenzamos BFS desde el centro, el árbol de búsqueda incorpora inmediatamente todos los vértices del ciclo exterior. El árbol resultante se conoce como estrella.

En cambio, si hacemos DFS desde el centro y luego avanzamos por el ciclo exterior, el árbol que se formará es un camino.

En los siguientes dibujos de $W_7$, $r$ es el centro y los vértices del ciclo exterior aparecen en el orden $a, b, c, d, e, f$. Las aristas continuas forman el árbol de búsqueda; las aristas grises discontinuas son las que no se eligen.




BFS: estrella.




DFS: camino.

### 4.4.4. Árboles generadores de costo mínimo

Hasta ahora cualquier árbol generador nos servía como certificado de conexidad. Si las aristas tienen costos distintos, tendremos que decidir cuál de esos árboles nos conviene construir. Durante esta subsección supondremos que aristas distintas tienen pesos distintos; al final discutiremos qué cambia cuando hay empates.

**Ejemplo.** Supongamos que tenemos ciudades $C_1, \ldots, C_n$ y queremos construir una red de carreteras sobre ellas. La red debería ser conexa, es decir, que desde cualquier ciudad se pueda llegar a cualquier otra. Además, sujeto a esta restricción, nos gustaría que el costo de construir la red sea lo más bajo posible.

Entre ciertos pares, digamos $C_i$ y $C_j$, podemos construir una carretera directa a un costo $w_{ij}$ dado. Representemos las ciudades por los vértices de un grafo y las carreteras que podemos construir por sus aristas. Supongamos que ese grafo es conexo y que los costos son positivos y distintos.

Si una red que conecta todas las ciudades contiene un ciclo, podemos quitar una de sus aristas y conservar la conexidad, como vimos en la clase anterior. Al hacerlo, el costo disminuye. Luego una solución óptima tiene que ser un árbol generador: entre todos esos árboles, buscamos el de menor costo total.

Para comparar los árboles, sumaremos los costos de sus aristas.

**Definición.** Un *grafo con pesos* es un grafo $G = (V, E)$ junto con una función $w\colon E \to \mathbb{R}$. Si $T$ es un subgrafo de $G$, su *peso* es $w(T) = \sum_{e \in E(T)} w(e)$.

En el ejemplo los costos eran positivos, lo que nos permitió reducir el problema de conectar las ciudades a elegir un árbol generador. Podemos estudiar esta última elección con pesos reales, manteniendo por ahora el supuesto de que son distintos: comparamos los pesos de todos los árboles generadores y buscamos el menor.

**Definición.** Sea $G$ un grafo simple con pesos $w$. Un *árbol generador de costo mínimo (MST)* de $G$ con pesos $w$ es un subgrafo $(U, T)$ tal que

- $(U, T)$ es un árbol generador,
- minimiza $\displaystyle\sum_{e \in T} w(e)$ entre todos los árboles generadores de $G$.

Antes de decidir cómo construir una solución, preguntémonos si puede haber más de una. El supuesto de pesos distintos nos permitirá descartar esa posibilidad mediante un intercambio de aristas.

**Teorema.** Sea $G$ un grafo simple, conexo y con pesos. Suponga que todos los pesos de las aristas son distintos. Entonces $G$ tiene un único árbol generador de costo mínimo.

*Demostración.* Como $G$ es conexo, tiene al menos un árbol generador. Hay finitos árboles generadores, así que alguno tiene costo mínimo. Supongamos que hay dos distintos, $T_1$ y $T_2$. Elijamos la arista $e$ de menor peso entre las que pertenecen a uno de ellos y no al otro. Intercambiando los nombres si es necesario, podemos suponer que $e \in E(T_1) \setminus E(T_2)$.

Al agregar $e$ a $T_2$ se forma un ciclo $C$. Como $T_1$ no contiene ciclos, alguna arista $f$ de $C$ no pertenece a $T_1$. Esa arista está en $E(T_2) \setminus E(T_1)$, y la elección de $e$ nos da $w(e) < w(f)$. Entonces $T_2 + e - f$ es un árbol generador y

$$w(T_2 + e - f) = w(T_2) + w(e) - w(f) < w(T_2),$$

contradiciendo que $T_2$ tiene costo mínimo. □

Volvamos a las ciudades del ejemplo. Si separamos un grupo de ciudades $U$ de las restantes, cualquier red que las conecte a todas debe usar alguna carretera entre ambos lados. ¿Podemos asegurar que la más barata de esas carreteras está en la solución óptima? La siguiente regla nos dice que sí.

**Lema** (Regla azul)**.** Sea $G = (V, E)$ un grafo simple, conexo y con pesos $w$ distintos, $U \subseteq V$. Si $e = uv$ es una arista de menor costo entre las aristas con $u \in U$ y $v \notin U$, entonces el MST la contiene.

*Demostración.* Sea $M$ el MST de $G$. Si $e \in E(M)$, entonces $M$ ya es un MST que contiene a $e$.

Supongamos entonces que $e \notin E(M)$. Al agregar $e$ a $M$, se forma un único ciclo. Como $e$ tiene un extremo en $U$ y el otro en $V \setminus U$, ese ciclo debe cruzar entre $U$ y $V \setminus U$ otra vez para volver al lado donde empezó. Por lo tanto, existe una arista $f \in E(M)$ en ese ciclo, distinta de $e$, que también tiene un extremo en $U$ y el otro en $V \setminus U$.

Como $e$ era de menor costo entre las aristas que cruzan de $U$ a $V \setminus U$ y los pesos son distintos,

$$w(e) < w(f).$$

Consideremos

$$M' = M + e - f.$$

Al borrar una arista de un ciclo, el grafo vuelve a ser un árbol generador. Además,

$$w(M') = w(M) + w(e) - w(f) < w(M).$$

Esto contradice que $M$ tiene costo mínimo. Por lo tanto, $e$ pertenece al MST. □

La regla azul sugiere cómo adaptar el algoritmo de exploración al problema de las carreteras. Partimos desde una ciudad y hacemos crecer el conjunto $U$ de ciudades ya conectadas. Una manera natural de adaptar este algoritmo para resolver el problema de árbol generador de costo mínimo es seleccionar la arista de menor peso que conecte un vértice en $U$ con un vértice en $V \setminus U$. Esto nos lleva al algoritmo de Prim, también llamado Prim–Jarník.

**Algoritmo 2** Prim

1: **Entrada:** un grafo simple y conexo $G = (V, E)$ con pesos distintos y una raíz $r \in V$.
2: **Salida:** un árbol generador de costo mínimo de $G$.
3: Partir con $U \leftarrow \{r\}$ y $T \leftarrow \emptyset$.
4: **while** $U \neq V$ **do**
5: Elegir $uv \in E$ con $u \in U$ y $v \notin U$ de menor costo.
6: $U \leftarrow U \cup \{v\}$, $T \leftarrow T \cup \{uv\}$.
7: **end while**
8: **return** $(U, T)$.

Ya sabemos que la exploración construye un árbol generador en un grafo conexo. La regla azul nos dice además que cada arista que elegimos debe pertenecer al MST.

> **Teorema.** Sea $G = (V, E)$ un grafo simple y conexo con pesos $w$ distintos. El algoritmo de Prim devuelve un árbol generador de costo mínimo.

*Demostración.* Primero veamos que Prim efectivamente construye un árbol generador. Por el lema del paso incremental, las aristas elegidas siempre forman un árbol sobre el conjunto $U$ de vértices ya incorporados. Como $G$ es conexo, mientras $U \neq V$ existe alguna arista con un extremo en $U$ y el otro en $V \setminus U$. Por lo tanto, Prim termina con un árbol generador.

Falta probar que ese árbol tiene costo mínimo. Sea $M$ el único MST de $G$. En cada paso, Prim elige la arista de menor costo entre el conjunto $U$ de vértices ya incorporados y el resto del grafo. Por la regla azul, esa arista pertenece a $M$. Luego todas las aristas del árbol construido están en $M$. Como ambos árboles generadores tienen $|V| - 1$ aristas, coinciden. Por lo tanto, Prim devuelve el MST. □

La regla azul nos dice cuándo una arista debe entrar en la solución. ¿Cuándo podemos asegurar que una arista debe quedar fuera? Volvamos a mirar los ciclos: si una carretera es la más cara de un ciclo, las otras ofrecen una forma de conectar sus extremos que permite evitarla.

> **Lema** (Regla roja)**.** Sea $G$ un grafo simple, conexo y con pesos $w$ distintos, y $C$ las aristas de un ciclo en $G$. Si $e = uv$ es una arista de $C$ de mayor costo, entonces no pertenece al MST.

*Demostración.* Sea $T$ el MST de $G$. Si $e \notin E(T)$, entonces $T$ ya es un MST que evita a $e$.

Supongamos que $e \in E(T)$. Al borrar $e$ de $T$, el árbol se separa en dos componentes conexas. Como las aristas de $C$ forman un ciclo, el resto de ese ciclo conecta los extremos de $e$. Al recorrerlo, necesariamente encontramos alguna arista $f \neq e$ que cruza entre las dos componentes que dejó $T - e$.

Entonces

$$T' = T - e + f$$

vuelve a ser un árbol generador. Como $e$ tiene mayor costo en $C$ y los pesos son distintos, tenemos $w(f) < w(e)$. Por lo tanto,

$$w(T') = w(T) - w(e) + w(f) < w(T).$$

Esto contradice la minimalidad de $T$. Luego $e$ no pertenece al MST. □

La regla roja nos permite construir el árbol descartando aristas que no pueden pertenecer a la solución óptima. Tratemos de diseñar un algoritmo glotón para resolver este problema. Una estrategia natural sería considerar las aristas en orden creciente de pesos y agregarlas si no generan conflictos. Es decir, si al agregar una arista no se forma un ciclo. Si una arista cierra un ciclo con las que ya elegimos, será la más cara de ese ciclo y podremos descartarla por la regla roja. Este es el algoritmo de Kruskal.

**Algoritmo 3** Kruskal

```
Entrada: un grafo simple y conexo G = (V, E) con pesos w distintos.
Salida: un árbol generador de costo mínimo de G.
Ordenar las aristas por peso w(e_1) < w(e_2) < ··· < w(e_m).
Partir con T ← ∅.
for i = 1, ..., m do
    if (V, T ∪ {e_i}) es acíclico then
        T ← T ∪ {e_i}.
    end if
end for
return (V, T).
```

La condición del algoritmo impide que aparezcan ciclos. Debemos justificar que logra conectar todos los vértices y que no descarta ninguna arista del MST; la regla roja nos permitirá probar esto último.

> **Teorema.** Sea $G = (V, E)$ un grafo simple y conexo con pesos $w$ distintos. El algoritmo de Kruskal devuelve un árbol generador de costo mínimo.

*Demostración.* Durante todo el algoritmo, el conjunto $A$ de aristas elegidas es acíclico, porque Kruskal sólo agrega una arista cuando no crea ciclos. Veamos que al final también conecta todos los vértices. Si quedaran dos componentes distintas en $(V, A)$, como $G$ es conexo habría alguna arista de $G$ que cruza entre ellas. Cuando Kruskal procesó esa arista, sus extremos estaban en componentes distintas, así que agregarla no habría creado un ciclo. Por lo tanto, Kruskal la habría agregado, contradiciendo que esas componentes quedaron separadas al final. Luego $(V, A)$ es conexo y acíclico, es decir, un árbol generador.

Probemos ahora que tiene costo mínimo. Sea $M$ el único MST de $G$. Si Kruskal descarta una arista $e$, es porque al agregarla al bosque actual se forma un ciclo. Todas las demás aristas de ese ciclo fueron consideradas antes, así que tienen menor peso que $e$. Por la regla roja, $e$ no pertenece a $M$. Luego ninguna arista de $M$ fue descartada, y $E(M) \subseteq A$. Como $(V, A)$ y $M$ son árboles generadores, ambos tienen $|V| - 1$ aristas y, por lo tanto, coinciden. □

### 4.4.5. Empates y otros algoritmos

Levantemos ahora el supuesto de que los pesos son distintos. Lo primero que puede perderse es la unicidad.

**Ejemplo.** Si las tres aristas de un triángulo tienen peso 1, cualquiera de sus tres árboles generadores tiene costo 2. Por lo tanto, los tres son MST.

Esto no impide que los algoritmos encuentren una solución óptima: basta decidir cómo romper los empates.

**Observación.** En un grafo simple y conexo con pesos que pueden repetirse, Prim y Kruskal siguen encontrando un MST al romper los empates arbitrariamente.

Para verlo, fijemos un orden arbitrario para desempatar y enumeremos las aristas como $e_1, \ldots, e_m$ siguiendo ese orden. Reemplacemos sus pesos por

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

Para $\varepsilon > 0$ suficientemente pequeño, estos pesos son distintos y no se invierte ninguna comparación estricta entre pesos de aristas ni entre costos de árboles generadores: sólo hay finitas aristas y finitos árboles generadores. Ejecutar Prim o Kruskal con estos pesos equivale a usar los pesos originales y el orden elegido para desempatar. Como el MST obtenido con los pesos perturbados también tiene costo mínimo con los originales, ambos algoritmos siguen funcionando. Si Prim desempata sobre la marcha, podemos reproducir sus elecciones ordenando primero sus aristas según el orden en que las agregó y después las restantes.

Las reglas azul y roja también se pueden combinar de otras maneras, dando lugar a algoritmos como *Round Robin*, *Borůvka* y *Kruskal inverso*. El desafío final propone estudiar el algoritmo de Borůvka.

# Ejercicios

1. *a*) Sea $G$ un grafo conexo y sea $T$ un árbol generador de $G$.
   1) Demuestre que si $e$ es una arista de $G$ que no pertenece a $T$, entonces $T + e$ contiene exactamente un ciclo.
   2) Demuestre que si $e$ es una arista de $T$, entonces $T - e$ tiene exactamente dos componentes conexas.

   *b*) Sea $G$ un grafo simple conexo. Diremos que una arista $e$ de $G$ es un *puente* si $G - e$ no es conexo. Demuestre que una arista $e$ pertenece a todo árbol generador de $G$ si y sólo si $e$ es un puente. Concluya que $G$ tiene un único árbol generador si y sólo si $G$ es un árbol.

   *c*) Un equipo quiere instalar túneles privados entre servicios de una aplicación distribuida. Los servicios son $A, B, C, D, E, F$ y los posibles túneles, junto con su costo mensual, son

| túnel | $AB$ | $AC$ | $BC$ | $BD$ | $CD$ | $CE$ | $DE$ | $DF$ | $EF$ | $BF$ |
|---|---|---|---|---|---|---|---|---|---|---|
| costo | 4 | 2 | 5 | 1 | 3 | 7 | 2 | 6 | 4 | 8 |

   Se quiere que todos los servicios puedan comunicarse entre sí, directamente o a través de otros servicios, minimizando el costo total.
   1) Modele el problema como un problema de árboles generadores de costo mínimo.
   2) Determine el costo mínimo total.

   *d*) Hay $n$ viviendas que se quieren conectar al suministro de agua. Entre las viviendas $i$ y $j$ se puede construir una cañería de costo $c_{ij}$. Además, la vivienda $i$ puede conectarse directamente a la matriz principal de agua con costo $C_i \geq 0$. Se quiere que todas las viviendas tengan acceso al agua al menor costo total posible. ¿Cómo puede resolver este problema usando algoritmos que ya conoce?

2. Alicia se desplaza por una ciudad que tiene lugares de interés etiquetados por $1, 2, \ldots, n$. Alicia sabe que en cada trayecto $ij$ hay un nivel de ruido $r_{ij}$. Suponga que $n \geq 2$, que estos niveles son positivos, que los trayectos se pueden recorrer en ambos sentidos y que forman un grafo simple y conexo. Alicia quiere seleccionar algunos de ellos para formar una red que conecte todos los lugares de interés.

   *a*) Considere una ciudad con seis lugares y únicamente los siguientes trayectos, con sus respectivos niveles de ruido:

| trayecto | 12 | 13 | 15 | 23 | 24 | 25 | 34 | 35 | 46 | 56 |
|---|---|---|---|---|---|---|---|---|---|---|
| ruido | 1 | 4 | 9 | 5 | 8 | 2 | 6 | 3 | 7 | 10 |

   Determine una red que conecte los seis lugares y minimice la suma de los niveles de ruido de sus trayectos. ¿Cuál es esa suma mínima?

   *b*) Sea $G$ un grafo simple, conexo y con pesos, y sea $T$ un MST de $G$. Escribamos $G = (V, E)$ y denotemos sus pesos por $w$. Para dos vértices $u, v \in V(G)$ distintos, sea $P_T(u, v)$ el único camino simple de $u$ a $v$ en $T$. Demuestre que $P_T(u, v)$ minimiza el peso de la arista más pesada del camino.

   Más precisamente, demuestre que para todo camino $P$ de $u$ a $v$ en $G$ se cumple

$$\max_{e \in P_T(u,v)} w(e) \leq \max_{e \in P} w(e).$$

*c*) ¿Cómo puede Alicia construir una red que conecte todos los lugares y minimice el máximo de los niveles de ruido de los trayectos elegidos? Describa y justifique un algoritmo usando la parte (b). En la instancia de la parte (a), indique una red óptima para este nuevo objetivo y el máximo nivel de ruido que alcanza.

3. Sea $G = (V, E)$ un grafo simple y conexo con pesos distintos, y sea $n = |V|$. El *algoritmo de Borůvka* comienza con $F = \varnothing$. Mientras $(V, F)$ no sea conexo, realiza una ronda de la siguiente forma:

   - Al comienzo de la ronda, calcula los conjuntos de vértices $C_1, \ldots, C_k$ de las componentes conexas de $(V, F)$.
   - Para cada $C_i$, elige la arista de menor peso con un extremo en $C_i$ y el otro fuera de $C_i$.
   - Agrega simultáneamente a $F$ el conjunto de aristas elegidas; si una arista fue elegida por dos componentes, la agrega una sola vez.

   *a*) Demuestre que, después de cada ronda, todas las aristas de $F$ pertenecen al único MST de $G$. Concluya que $(V, F)$ es un bosque.

   *b*) Demuestre que una ronda que comienza con $k > 1$ componentes termina con a lo más $k/2$ componentes.

   *c*) Concluya que el algoritmo termina en a lo más $\lceil \log_2 n \rceil$ rondas y devuelve el MST de $G$.