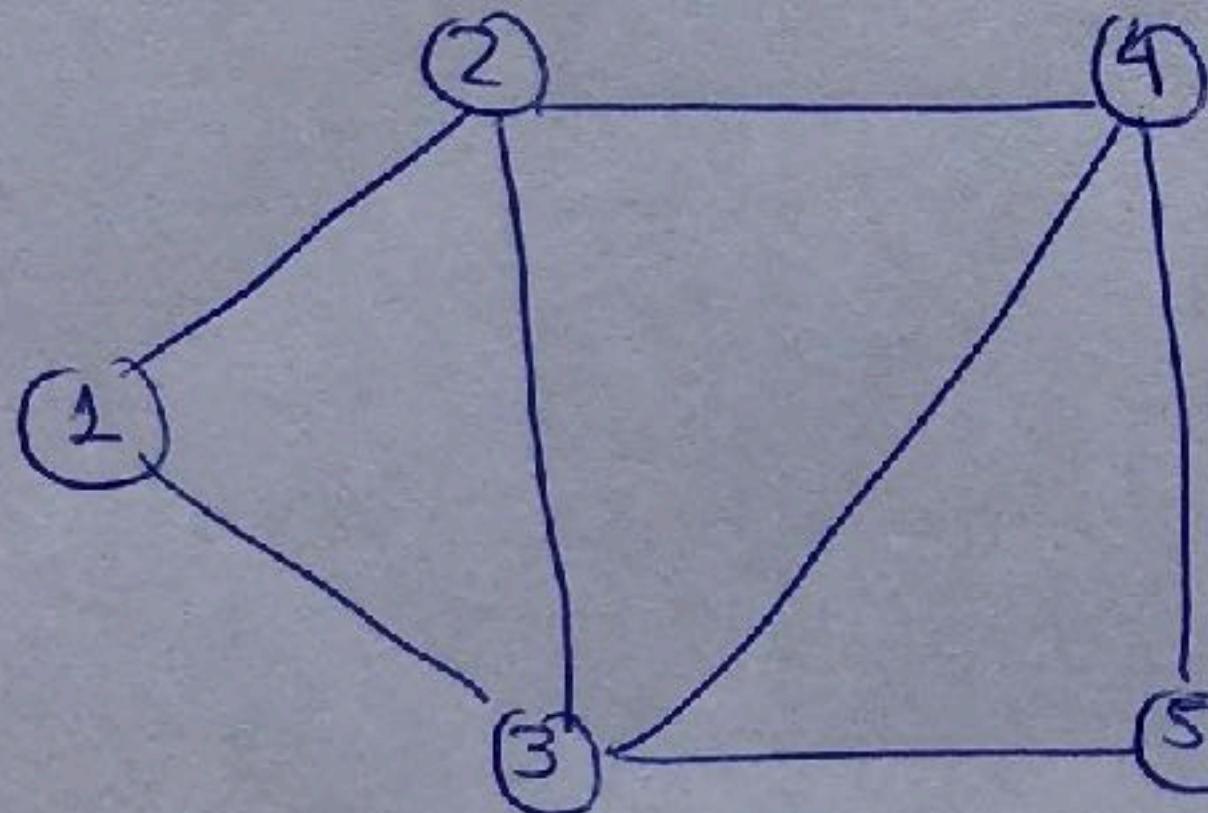


1. ÁRBOLES GENERADORES DE PESO MÍNIMO

1.1. DEFINICIONES BÁSICAS

Un grafo es un par ordenado $G = (V, E)$ donde

- V es un conjunto finito de nodos o vértices [nodes]
- E es un multiconjunto de aristas, donde una [edges] arista es un par (no ordenado) de nodos.



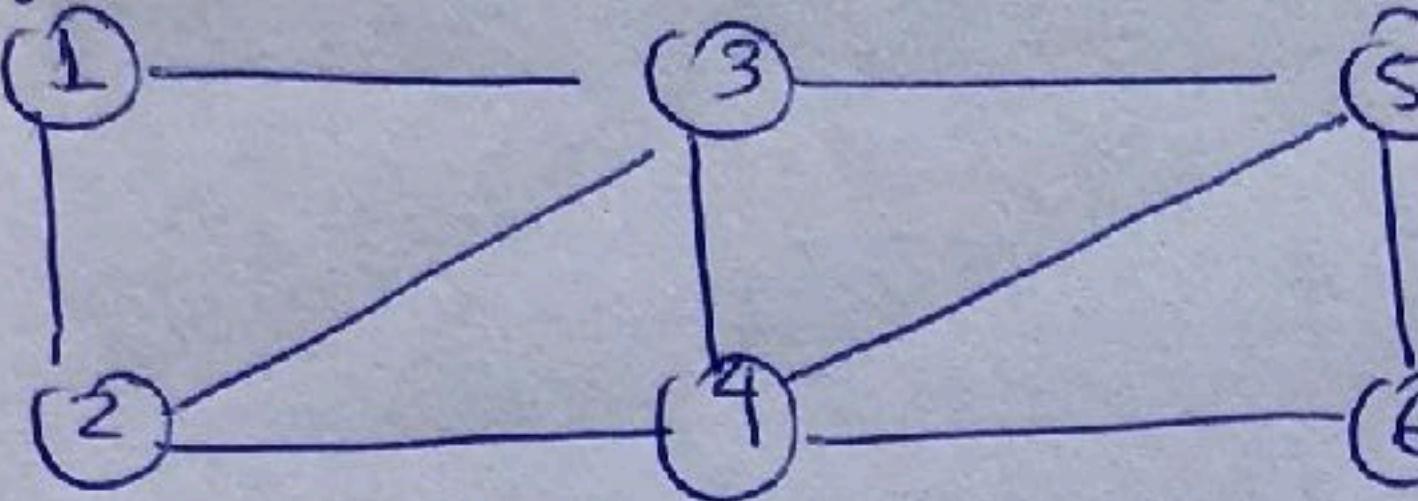
$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$$

Notación: Si no hay lugar a confusión, usaremos i, j para denotar a la arista $\{i, j\}$.

- Si $e=ij$ es una arista, los nodos i y j se llaman extremos de e .
- Una arista e es incidente a un nodo i , si i es un extremo de e .
- El conjunto de aristas incidentes a un nodo i se denota por $d(i)$.
- El grado de un nodo i se define por $d(i) := |\overrightarrow{d(i)}|$
- Dos nodos $i, j \in V$ se llaman adyacentes o vecinos, si $ij \in E$.
- La vecindad de un nodo i es el conjunto de todos los nodos que son vecinos a i , y se denota por $N(i)$.

Ejemplo:



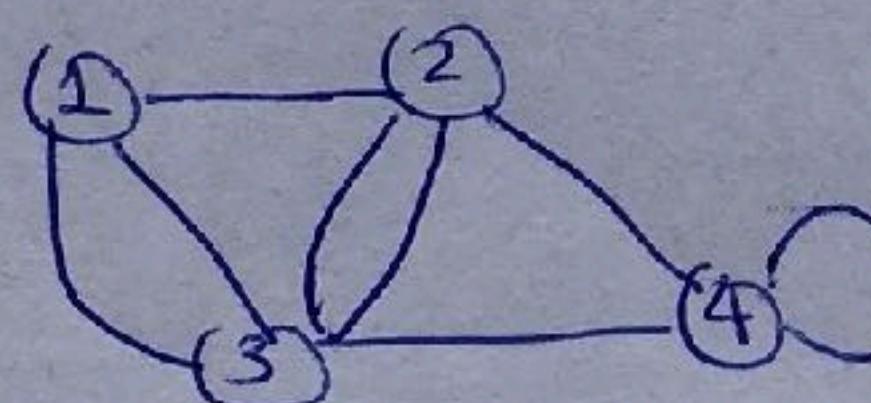
$$N(1) =$$

~~$$N(6) =$$~~

$$\delta(4) =$$

$$d(5) =$$

- Dos aristas se llaman paralelas, si tienen los mismos extremos
- Una arista es un lazo si tiene la forma $e=ii$, para algún $i \in V$.
- Un grafo sin aristas paralelas ni lazos se llama grafo simple



Nota: A menos que se indique lo contrario, en adelante asumiremos que trabajamos sobre grafos simples.

Teorema 1: Sea G un grafo sin lazos (aunque puede contener aristas paralelas). Demostrar que la suma de los grados de los nodos de G es una cantidad par.

[Ejercicio].

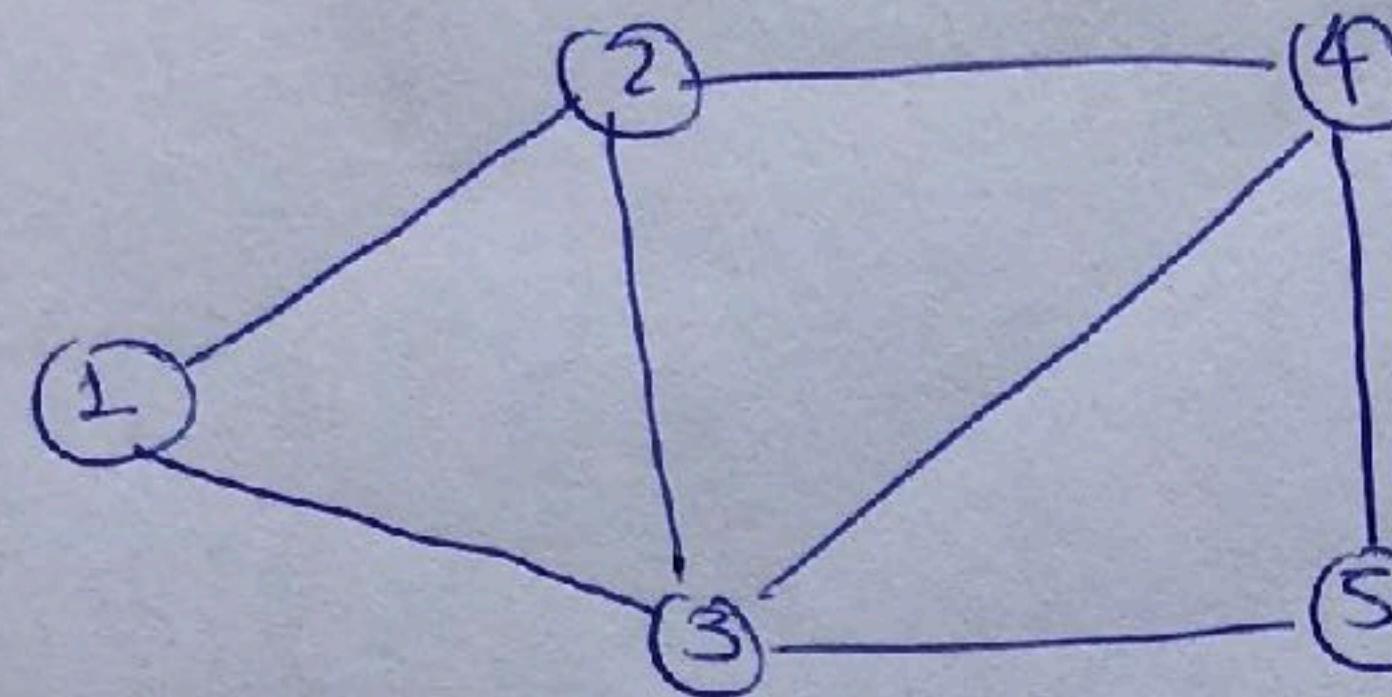
Definición (Sendero)

Un sendero [path] es una sucesión alternada de nodos y aristas de la forma

$P: i_0, e_1, i_1, e_2, i_2, \dots, e_k, i_k$

Donde: $\begin{cases} i_0, i_1, \dots, i_k \in V \\ e_1, e_2, \dots, e_k \in E \\ e_l \text{ tiene como extremos a } i_{l-1}, i_l, \forall l \in \{1, \dots, k\} \end{cases}$

Un sendero representa una trayectoria dentro de G:



P: 2, 23, 3, 13, 1, 12, 2, 23, 3, 35, 5

Un sendero es simple si no contiene aristas repetidas.

Un sendero es elemental si no contiene nodos repetidos, con excepción (posiblemente) de $i_0 = i_k$.

Un sendero es cerrado, si $i_0 = i_k$. [closed path]

Un sendero elemental cerrado se llama ciclo. [cycle]

Ejercicio: Identificar ejemplos de estos tipos de senderos en el grafo anterior.

Definición (Subgrafo)

Dado un grafo $G = (V, E)$, un subgrafo de G es un nuevo grafo $G' = (V', E')$ tal que $V' \subseteq V$ y $E' \subseteq E$.

Obs: Es necesario que (V', E') sea un grafo. Esto NO se cumple en general para cualquier $V' \subseteq V$ y $E' \subseteq E$.

- Si $V' = V$, entonces (V', E') es un subgrafo de G para cualquier $E' \subseteq E$. Un subgrafo de esta forma se llama subgrafo generador. [spanning subgraph]

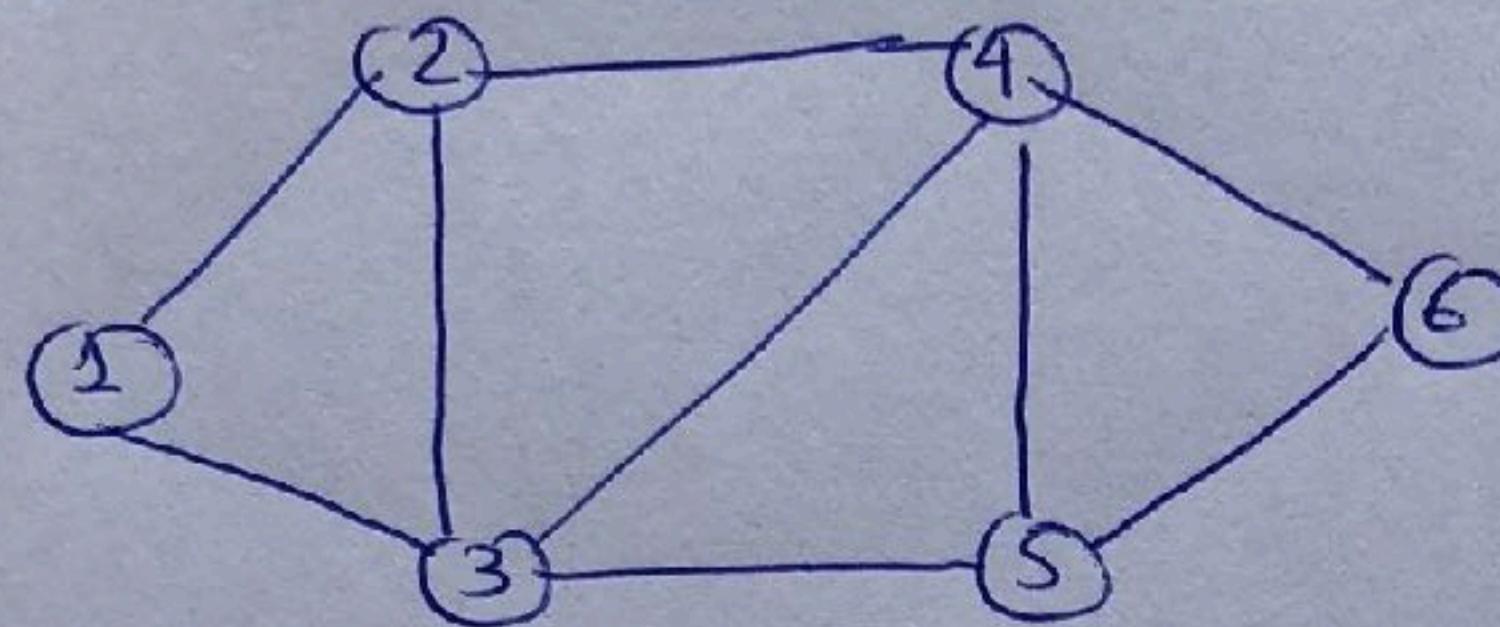
- Si $V' \subseteq V$ y $E' = E[V']$ donde

$$E[V'] := \{ ij \in E \mid i \in V' \wedge j \in V' \}$$

es el conjunto de amistas internas a V' , entonces (V', E') es subgrafo de G . Un subgrafo de esta forma se llama subgrafo inducido por V' . [induced subgraph]

Ejercicio:

$G:$



$$V_1 = \{1, 2, 3\}$$

(V_1, E_1) es subgrafo de G ?

$$E_1 = \{12, 23\}$$

(V_1, E_2) es subgrafo de G ?

$$E_2 = \{12, 34, 45\}$$

Dar ejemplos de subgrafos generadores y subgrafos inducidos de G .

Definición (Conexidad)

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

Sean $i, j \in V$. Decimos que i está conectado con j .

Si existe un sendero en G que tiene a i como nodo inicial y a j como nodo final.

(Notar que i está conectado con $j \Leftrightarrow j$ está conectado con i)

Decimos que G es un grafo conexo [connected graph]
si cada par de nodos están conectados entre sí.

Notar que la relación "estar conectado con" es una relación de equivalencia sobre el conjunto de nodos (asumimos que $P: i_0, \text{ con } i_0 \in V, \text{ es un sendero}$).
[Ejercicio]

Luego, la relación "estar conectado con" induce una partición de V en clases de equivalencia.

S_1, \dots, S_k :

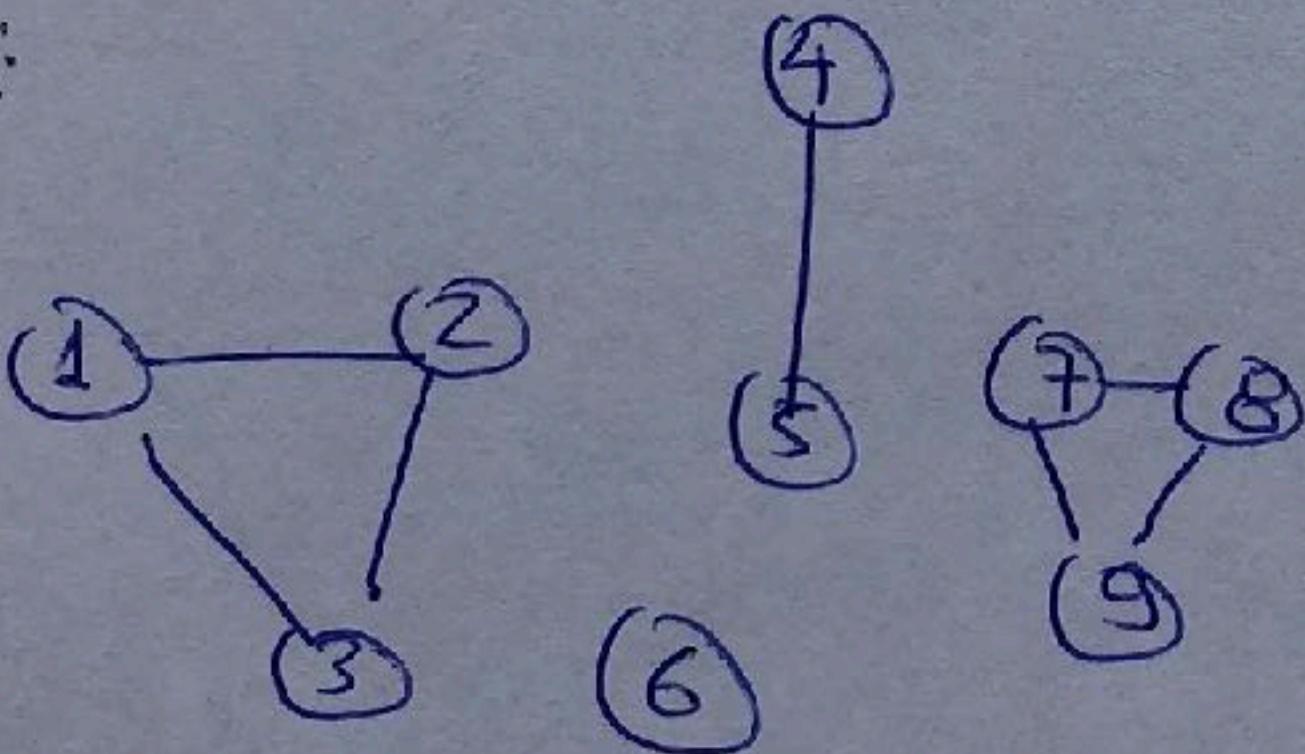
$$V = \bigcup_{i=1}^k S_i$$

donde $G[S_i]$ es un grafo conexo, $\forall i=1, \dots, k$.

Estos grafos se llaman componentes conexas de G .

En particular, si G es conexo, G admite una única componente conexa.

Ejemplo:

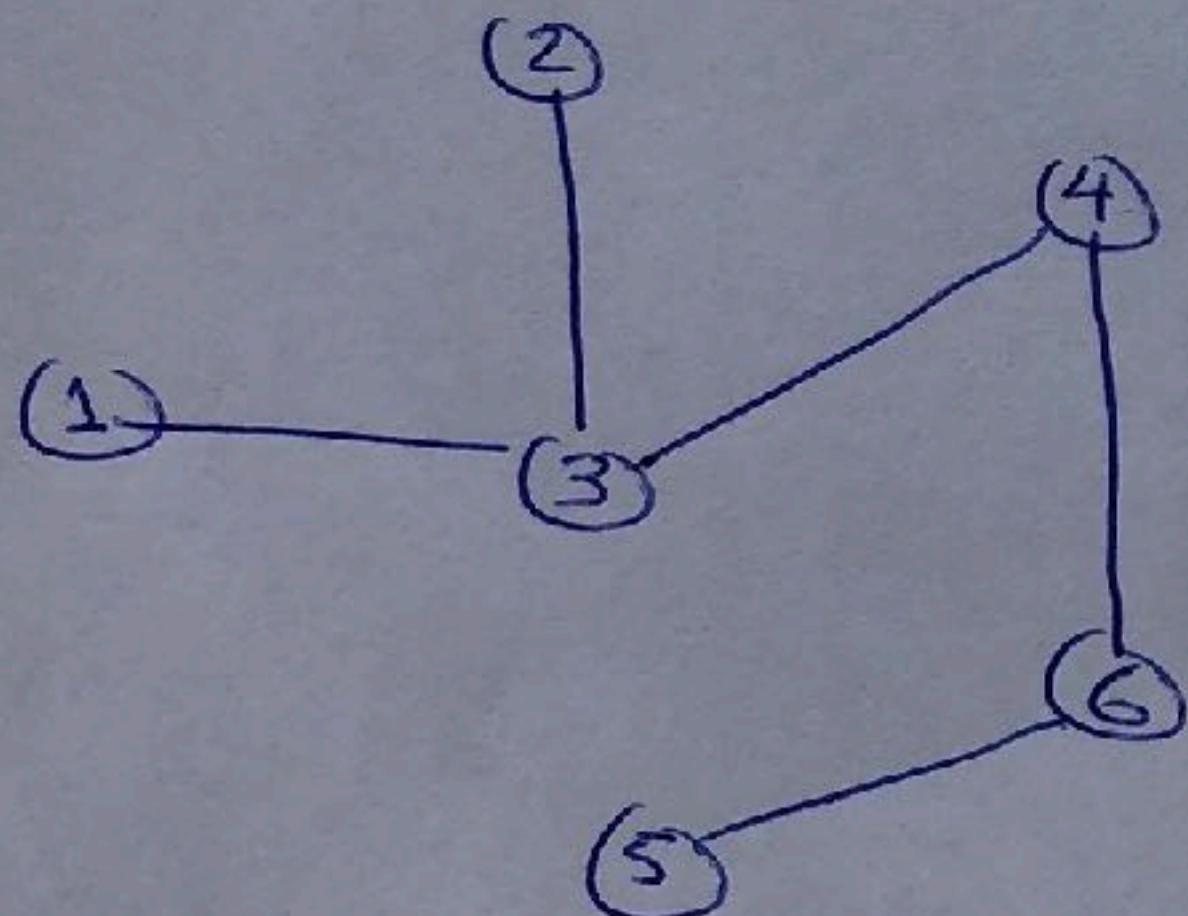


1.2. Árboles generadores de peso mínimo

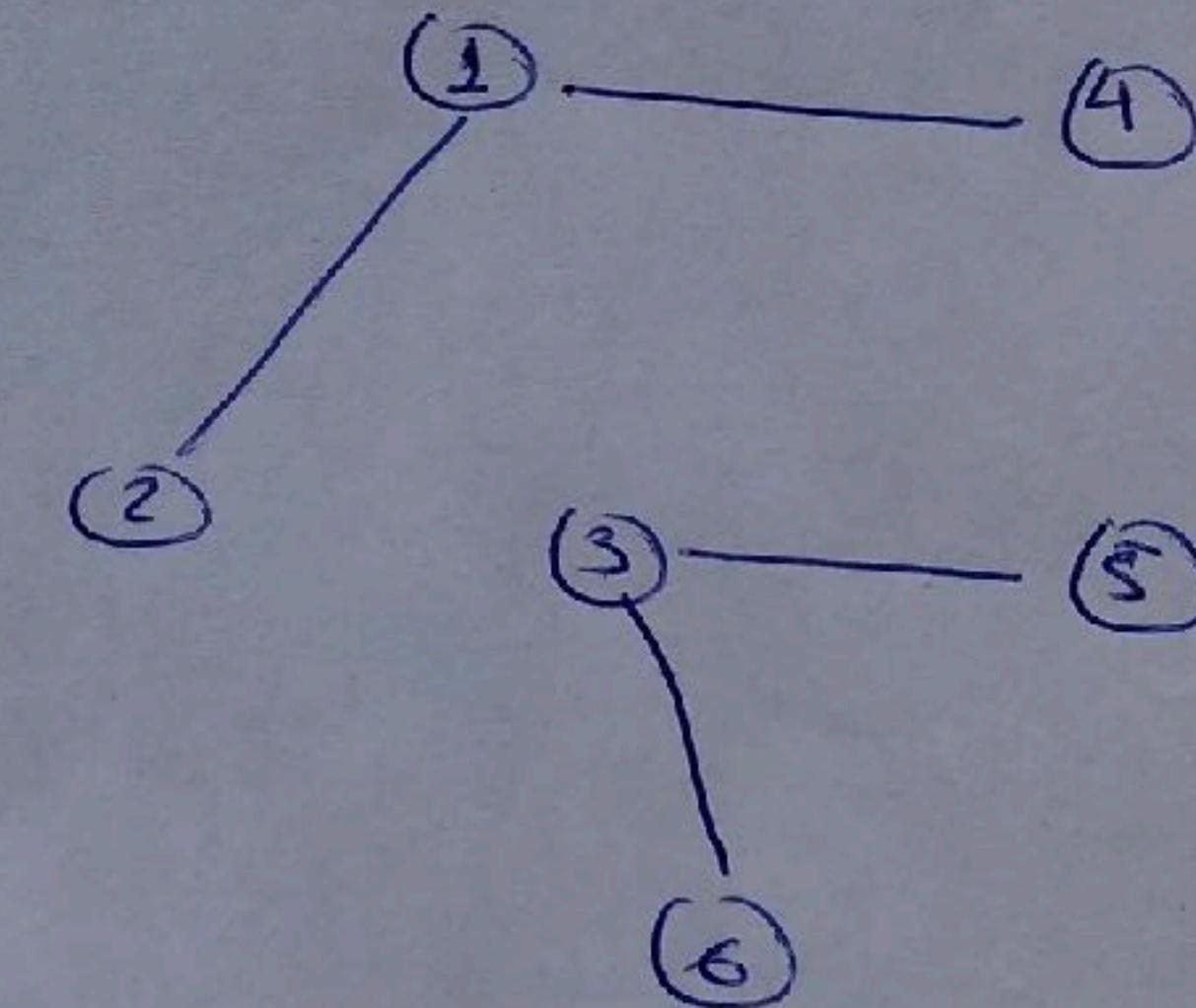
Definición: Un árbol es un grafo conexo y sin ciclos.

Un bosque es un grafo sin ciclos, como subgrafos (cada componente conexa de un bosque es un árbol).

Ejemplos:



⋮



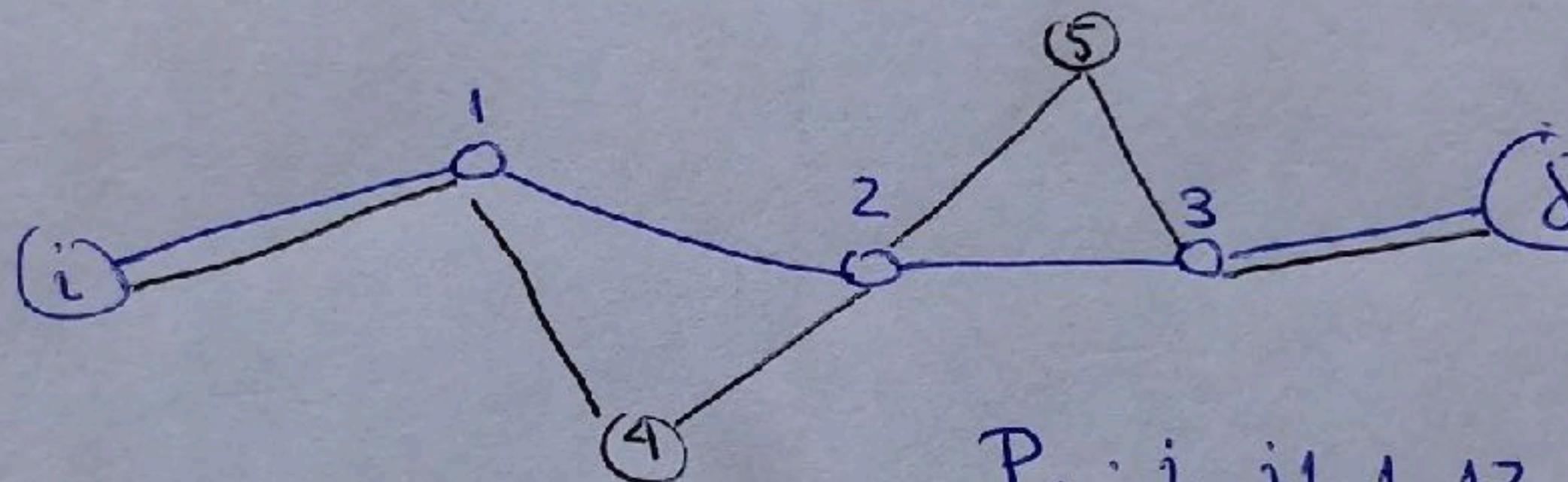
Proposición - 1.2. - (Propiedades de los árboles)

Sea $G = (V, T)$ un árbol.

- (i) $\forall i, j \in V$, existe un único sendero ^{simple} que conecta i con j .
- (ii) $\forall e \in T$; el grafo $(V, T \setminus \{e\})$ no es conexo.
- (iii) $\forall \hat{e} \in \binom{V}{2} \setminus T$, el grafo $(V, T \cup \{\hat{e}\})$ contiene un círculo.
pares de nodos
- (iv) $|T| = |V| - 1$.

Demonstración.

(i) Sean $i, j \in V$. Como G es conexo, existe al menos un sendero que conecta i con j . Por otra parte, si existen dos o más senderos ^{simples} distintos que conectan i con j , entonces G contiene un ciclo.



$$P_1: i, i1, 1, 12, 2, 23, 3, 3j, j$$

$$P_2: i, i1, 1, 14, 4, 24, 2, 25, 5, 35, 3, 3j, j$$

- (ii) { (Ejercicio)
(iii)

(iv) Si $|V|=1$, el único árbol de un nodo es el grafo $G=(\{1\}, \emptyset)$ que contiene un nodo aislado.

①

Por inducción, supongamos que $|T|=|V|-1$ se cumple para todos los árboles con $|V|\leq k$ nodos, para un cierto $k \in \mathbb{N}$.

Sea $\hat{G}=(\hat{V}, \hat{T})$ un árbol, con $|\hat{V}|=k+1$.

Notar que debe existir $i \in \hat{V}$ t.q. $d(i)=1$. (Un nodo con grado 1 se llama "hoja" del árbol).

[Ejercicio]

Sea \tilde{G} el subgrafo de \hat{G} inducido por $\hat{V} \setminus \{i\}$.

Notar que \tilde{G} no tiene ciclos, pues \hat{G} es un árbol.

Además, para $j, k \in \hat{V} \setminus \{i\}$, se cumple que el sendero que conecta j con k en \hat{G} no contiene a i . (Por qué?)

Luego, el mismo sendero conecta j con k en \tilde{G} .

Por lo tanto, \tilde{G} es conexo.

$\Rightarrow \tilde{G}$ es un árbol.

Además, \tilde{G} contiene un nodo y una arista menos que \hat{G} .

Es decir, \tilde{G} contiene k nodos y, por hipótesis de inducción, $k-1$ aristas.

Luego, \hat{G} contiene $|V| = k+1$ nodos y $|T| = k$ aristas.

Problema: Árbol generador de peso mínimo

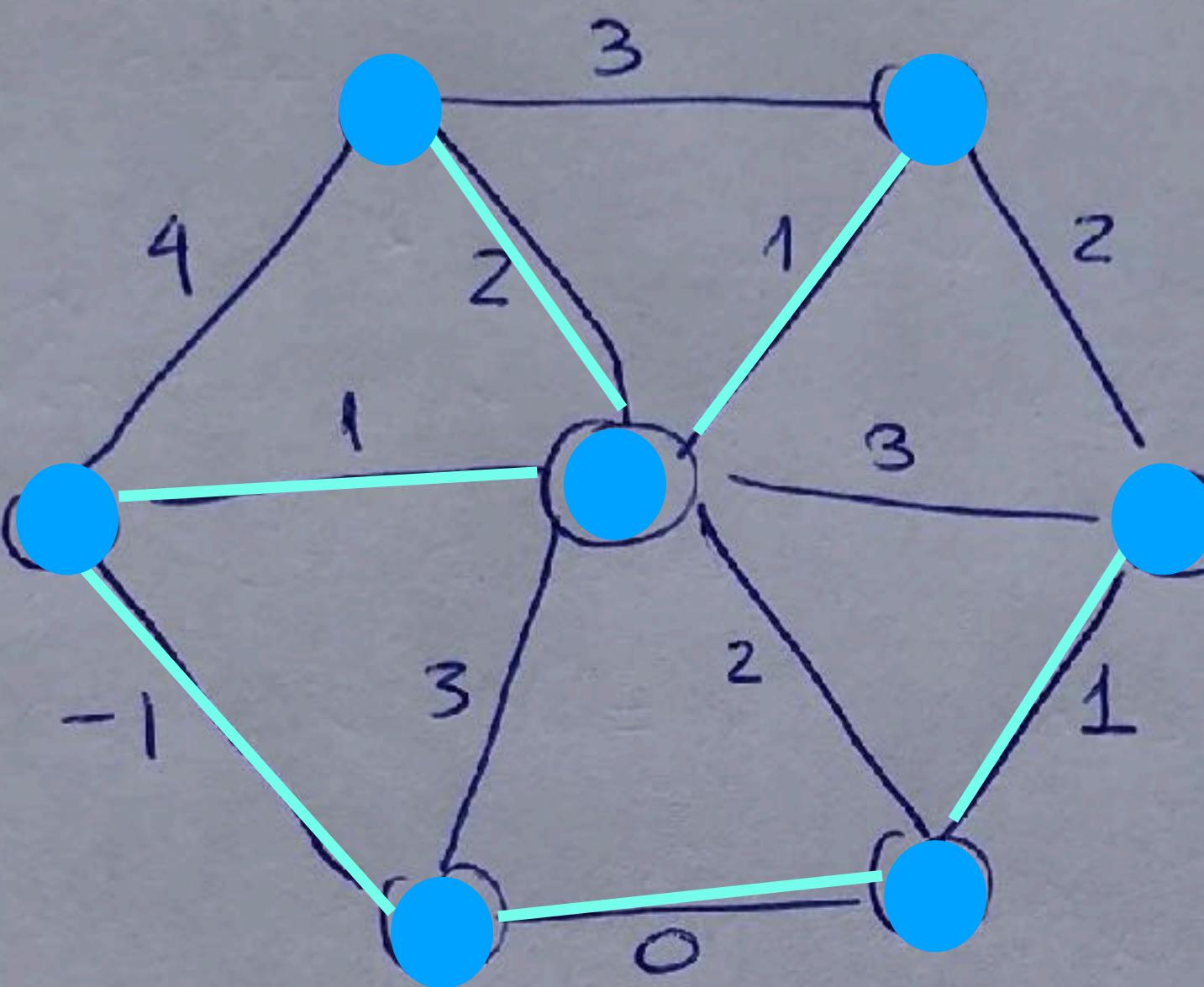
[Minimum Spanning Tree - MST]

Dados un grafo conexo $G = (V, E)$ y un vector de costos $c \in \mathbb{R}^E$ asociado a las aristas de G , encontrar $T \subseteq E$ que satisfaga

- $H := (V, T)$ es un árbol
- $c(H) := \sum_{e \in T} c_e$ es mínimo

Ejemplo :

Encontrar un MST por inspección



$$c(H) = 4$$

¿ Cómo hacerlo sistemáticamente ?

Algoritmo de Prim:

Seleccionar $i \in V$ arbitrario.

Fijar $W := \{i\}$
 $T := \emptyset$

Mientras $W \neq V$, hacer:

- Seleccionar $e = ij \in \delta(W)$ de costo mínimo
- Suponer $i \in W, j \notin W$. Hacer:
 $W := W \cup \{j\}$
 $T := T \cup \{e\}$

Notación: $\delta(W) := \{e = ij \in E \mid i \in W, j \notin W\}$

es el corte de W .

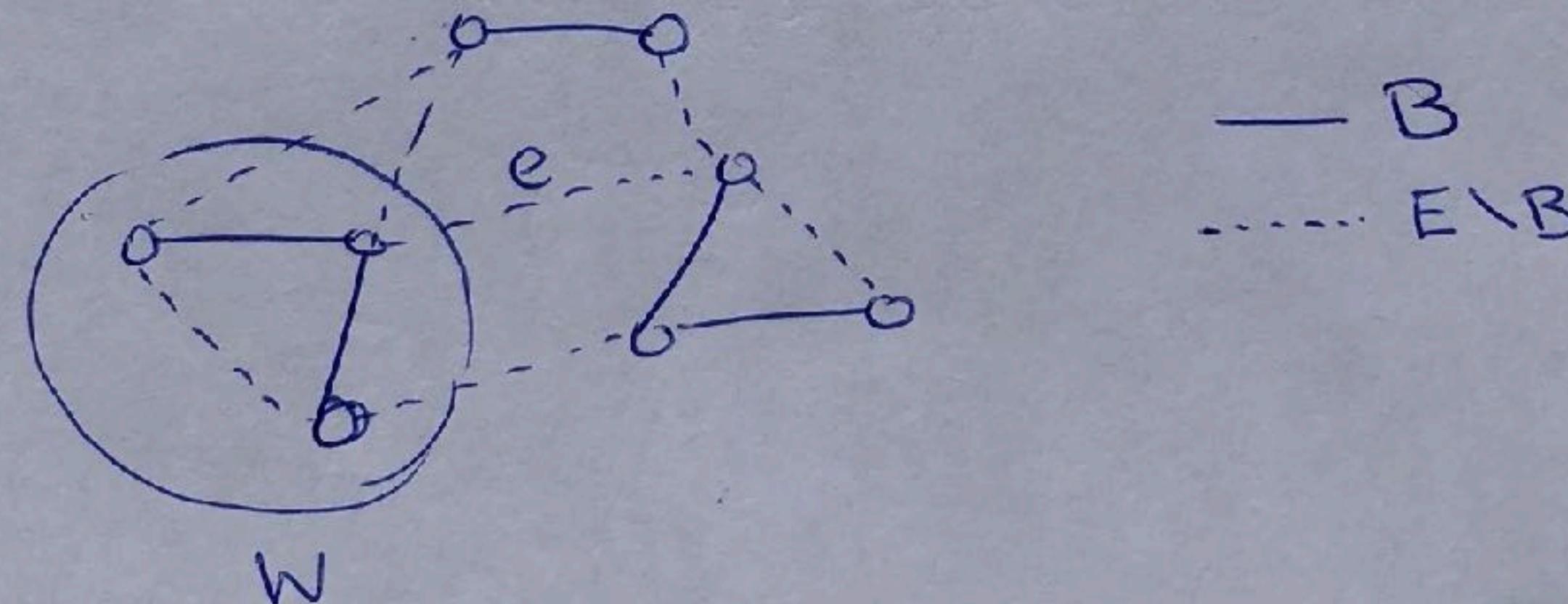
$E(W) := \{e = ij \in E \mid i \in W, j \in W\}$ son las arestas
internas de W .

Definición (Conjunto extensible)

Sea $G = (V, E)$ un grafo conexo. Un conjunto $B \subseteq E$ de aristas se llama extensible, si existe un árbol generador de peso mínimo $H = (V, T)$ tal que $B \subseteq T$.

Lema 1.3.

Sean $B \subseteq E$ un conjunto extensible y $W \subseteq V$ t.q.
 $\delta(W) \cap B = \emptyset$. Si e es una arista de costo mínimo
de $\delta(W)$, entonces $B \cup \{e\}$ es extensible.



Demostración:

Como B es extensible, existe un árbol generador de peso mínimo $H = (V, T)$ tal que $B \subseteq T$.

Si $e \notin T$, no hay nada que demostrar, pues $B \cup \{e\} \subseteq T$.

Caso contrario, suponer que $e = ij$, con $i \in W$ y $j \notin W$.

Como H es un árbol, existe un sendero único que conecta i con j usando aristas de T . Notar que $e \notin T$ y luego este sendero no contiene a e .

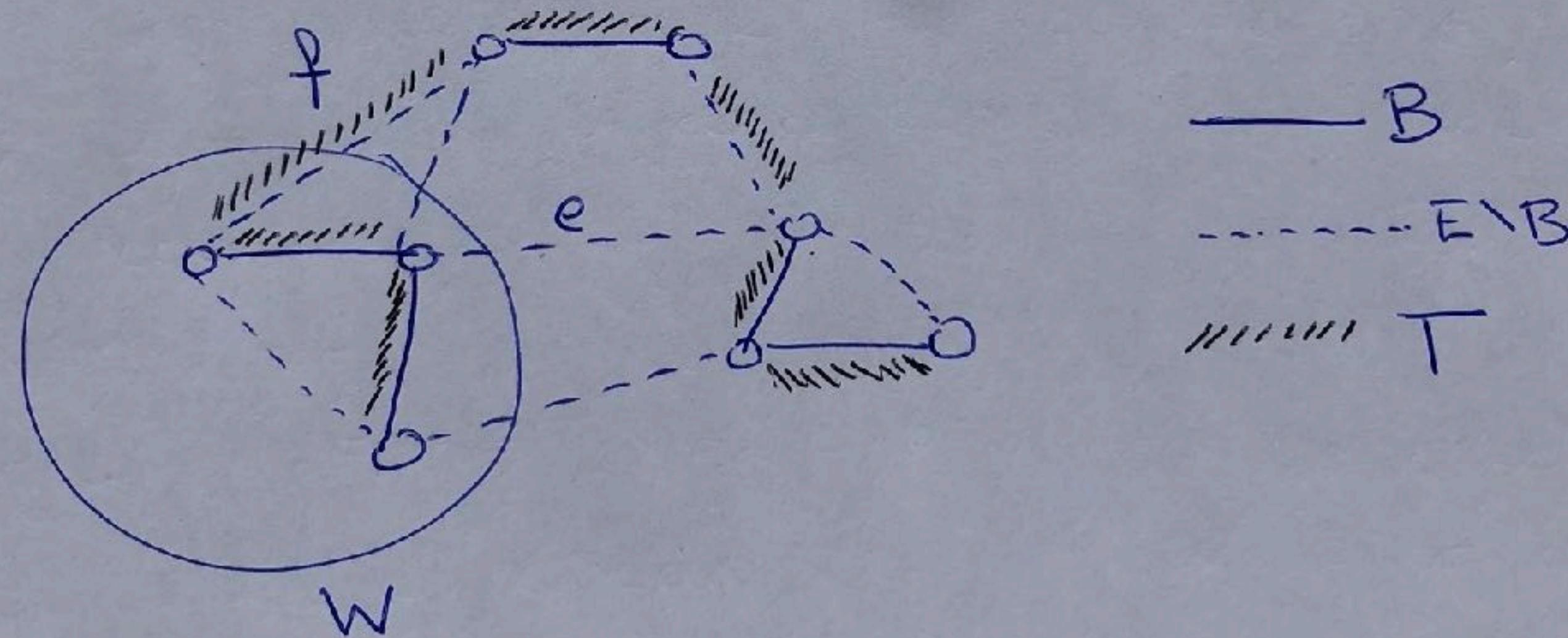
Por otra parte, este sendero conecta un nodo en W con un nodo afuera de W , por lo que debe contener una arista $f \in \delta(W)$.

Notar que $\hat{H} := (V, T \setminus \{f\} \cup \{e\})$ es un árbol generador, pues al añadir e a H se forma un único ciclo que contiene a f .

Finalmente, como $c_e \leq c_f$ (Por qué?),
se tiene $c(\hat{H}) = c(H) + c_e - c_f \leq c(H)$.

Pero H es de peso mínimo, luego $c(H) \leq c(\hat{H})$ y
por tanto $c(\hat{H}) = c(H)$, es decir, \hat{H} es también un
árbol generador de peso mínimo.

$\Rightarrow B \cup \{e\}$ es extensible.



Teorema 1.4.

El algoritmo de Prim calcula un árbol generador de peso mínimo.

Demostración:

- Notar que el lazo principal del algoritmo se ejecuta un número finito de veces, pues el conjunto W "crece" en un nodo nuevo en cada iteración
- Al finalizar cada iteración del lazo principal, se tiene que (W, T) es un árbol. (Por qué?)
- Por lo tanto, al finalizar el algoritmo (W, T) es un árbol generador, pues $W = V$.
- Al iniciar el algoritmo, $T = \emptyset$ es extensible.
- En cada iteración del algoritmo se añade a T

una arista e de costo mínimo de $\delta(W)$.

Como $T \subseteq E(W)$, entonces $T \cap \delta(W) = \emptyset$ y aplicando el Lema 1.3, se sigue que $T \cup \{e\}$ es extensible.

$\Rightarrow T$ es un conjunto extensible al finalizar cada iteración del algoritmo

Por lo tanto, al finalizar el algoritmo (W, T) es un árbol generador de peso mínimo.

Eficiencia de un algoritmo.

¿ Cuánto demora un algoritmo en resolver
un problema ?

¿ Cuándo consideramos que un algoritmo es
rápido ?

Observaciones :

- El tiempo de ejecución de un algoritmo depende (además de su implementación) del compilador en el que es ejecutado.
- El tiempo depende también del tamaño de la instancia.

Por lo tanto...

- En lugar del tiempo de ejecución, nos interesa medir el número de operaciones que requiere un algoritmo para resolver un problema.
- Nos interesa expresar este número de operaciones como una función del tamaño de las instancias. Esta función suele denominarse complejidad temporal del algoritmo y se define por:

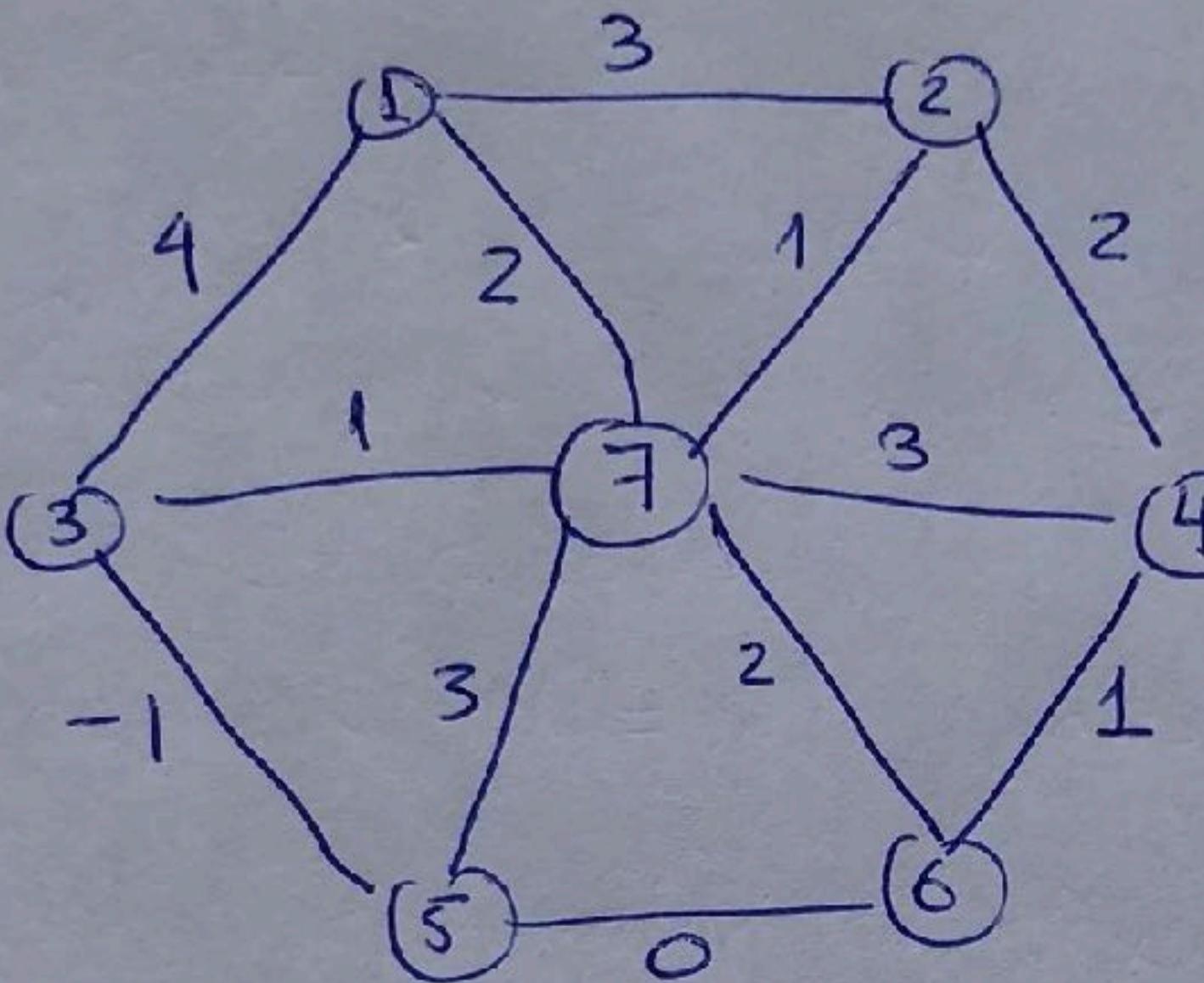
$$T: \mathbb{N} \longrightarrow \mathbb{N}$$

$$n \mapsto T(n) := \sup \left\{ t(I) \mid I \in \mathcal{I}, |I| \leq n \right\}.$$

Donde:
 \mathcal{I} : es el conjunto de instancias del problema.
 $|I|$: es el tamaño de la instancia $I \in \mathcal{I}$
 $t(I)$: es el número de operaciones que el algoritmo requiere para resolver $I \in \mathcal{I}$.

Ejemplo :

Encontrar un MST por inspección



¿ Cómo hacerlo sistemáticamente ?

$$I = (1,3,4)(3,5, -1)(3,7,1)(1,7,2)(1,2,3)(2,4,2)(4,6,1)(5,6,0)(2,7,1)(4,7,1)(5,7,3)(6,7,2)$$

$$\langle I \rangle = 85$$

Determinar $\langle I \rangle$, $t(I)$ y T con exactitud es imposible para la mayoría de casos reales.

Suele utilizarse la notación asintótica:

Definición:

Sean f, g dos funciones $\mathbb{R} \rightarrow \mathbb{R}$. Decimos que:

- (i) $f \in O(g)$ si $\exists x_0 \in \mathbb{R}, c \in \mathbb{R}_+$ t.q. $f(x) \leq c \cdot g(x), \forall x \geq x_0$
- (ii) $f \in \Omega(g)$ si $\exists x_0 \in \mathbb{R}, c \in \mathbb{R}_+$ t.q. $f(x) \geq c \cdot g(x), \forall x \geq x_0$
- (iii) $f \in \sigma(g)$ si $f \in O(g) \wedge f \in \Omega(g)$.

Ejemplos:

1)	$x^2 + 100x + 3 \in O(x^2)$	$\in \Omega(x^2)$	$\in \sigma(x^2)$
	$\in O(x^3)$	$\notin \Omega(x^3)$	$\notin \sigma(x^3)$
	$\notin O(x)$	$\in \Omega(x)$	$\notin \sigma(x)$

$$2) \quad \ln(x) \in O(\log(x)) \\ \in \Omega(\log(x)) \\ \in \Theta(\log(x))$$

$$3) \quad \log_a(x) \in O(\log_b(x)) \\ \forall a, b \in \mathbb{N} \setminus \{1\}$$

$$4) \quad x^n \in O(2^x), \forall n \in \mathbb{N} \\ \notin \Omega(2^x) \\ \notin \Theta(2^x)$$

$$5) \quad 2^x \in O(3^x) \\ \notin \Omega(3^x) \\ \notin \Theta(3^x).$$

$$6) \quad O(1) \subset O(\log x) \subset O(x) \subset O(x^2) \subset O(x^n) \subset O(2^x)$$

\downarrow \downarrow \downarrow \downarrow \downarrow
 constante logarítmica lineal cuadrática polinomial exponencial

----- ¿Cómo medir el tamaño de una instancia?

El tamaño de una instancia $I \in \mathcal{I}$ es la cantidad de caracteres de un cierto alfabeto Σ requeridas para expresar de manera inequívoca toda la información de entrada asociada a la instancia. Denotaremos por $\langle I \rangle$ al tamaño de la instancia.

- En el modelo antimétrico, asumimos que cada número puede representarse con un número constante de caracteres.
- En el modelo de bit, debemos estimar la cantidad de caracteres necesaria para representar cada número del problema

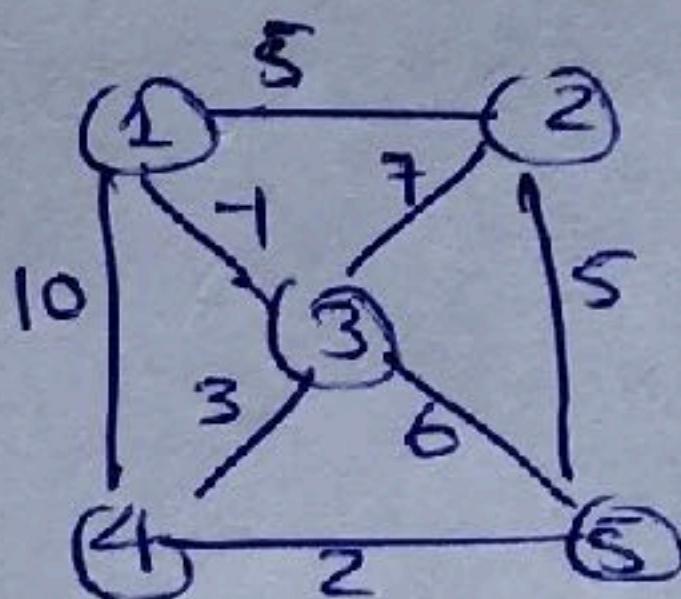
Ejemplo:

Para el problema de árboles generadores de peso mínimo, cada instancia consiste de un grafo $G = (V, E)$ y un vector $c \in \mathbb{R}^E$. ¿Cuál es el tamaño de una instancia para un grafo con n nodos y m aristas?

Representación:

Usando el alfabeto $\Sigma = \{(,), 1, 0, \dots, 9\}$ podemos representar cualquier instancia enumerando la información para cada arista en la forma $(i|j|c_{ij})$.

Ejemplo:



\Rightarrow

$$\begin{aligned}(1|2|5) & (1|3|10) \\ (2|5|5) & (2|3|7) \\ (3|5|6) & (4|5|2)\end{aligned}$$

$$\langle I \rangle = m(4 + 3K)$$

Longitud de codificación:

Modelo aritmético:

$$\langle I \rangle = 7m \in \sigma(m).$$

Modelo de bit:

$$\langle I \rangle = 4m + \sum_{ij \in E} (\langle i \rangle + \langle j \rangle + \langle c_{ij} \rangle)$$

asumiendo
costos
enteros,

$$\underbrace{\langle I \rangle}_{\leq} = 4m + \sum_{ij \in E} (\lceil \log(n) \rceil + \lceil \log(n) \rceil + \lceil \log(k) + 1 \rceil)$$

$$y c_{ij} \leq k, \forall ij \in E$$

$$\in O(m + m \cdot \log(n) \cdot \log(k))$$

$$\subseteq O(m \cdot \log(n) \cdot \log(k))$$

¿Cuándo consideramos a un algoritmo "rápido" o "eficiente?"

- Depende del propósito y del contexto práctico.
- En la Teoría de Complejidad Computacional, un algoritmo es eficiente si es polinomial, es decir, si

$$\exists k \in \mathbb{N} : T \in O(n^k)$$

En otras palabras, si existe un polinomio p con la propiedad de que el número de operaciones requeridas por el algoritmo para resolver una instancia $I \in \mathcal{I}$ del problema está acotado superiormente por $p(\langle I \rangle)$.

Clases de problemas (una aproximación)

- Clase P: Problemas para los cuales existe un algoritmo polinomial (círculos generadores de peso mínimo, caminos más cortos, flujos máximos, flujos de costo mínimo, ...)
- Clase NP: Problemas para los cuales la validez de una solución dada puede ser verificada por un algoritmo polinomial (casi todos los problemas conocidos de la optimización combinatoria).
- Problemos NP-difíciles: La existencia de un algoritmo polinomial para cualquier problema de esta clase implica que $P=NP$.
(Problema del agente viajero, coloración en grafos, circuitos hamiltonianos, árboles de Steiner, ...)

Complejidad del algoritmo de Prim

- Elegir i_0 arbitrariamente
 - $W := \{i_0\}$
 - $T := \emptyset$
- $\} O(1)$

Mientras $W \neq V$

- $n-1$
veces
- Elegir $e = ij \in \delta(W)$ de costo mínimo $\} O(m)$
 - Asumiendo $i \in W, j \notin W$, hacer:
 - $W := W \cup \{j\}$
 - $T := T \cup \{e\}$ $\} O(1)$

- Una implementación sencilla del algoritmo puede mantener un vector indicativo de W y recorrer la lista de todas las aristas para verificar si pertenecen a $\delta(W)$ y cuál es la de costo mínimo. Esto requiere $O(m^2)$ operaciones.

- Como el lazo principal se repite $n-1$ veces, el algoritmo requiere en total $O(m \cdot n)$ operaciones.
- Para un grafo simple y conexo, se cumple:

$$n-1 \leq m \leq \frac{1}{2}n(n-1)$$

Luego, $O(m \cdot n) < O(m^2)$

y se concluye que el algoritmo de Prim es polinomial.

- Si se almacena para cada $j \in V \setminus W$ la arista $e(j)$ de costo mínimo que conecta a j con algún nodo en W , entonces elegir la arista de costo mínimo de $\delta(W)$ puede hacerse comparando los valores de $e(j)$, para $j \in V \setminus W$, lo que

requiere de $O(n)$ operaciones.

Cada vez que se cambia W al finalizar una iteración, es necesario verificar y actualizar los valores de $e(j)$. Sin embargo, esto puede hacerse también en $O(n)$ operaciones.

- Con esta mejora en la implementación, el algoritmo de Prim requiere $O(n^2)$ operaciones.

1.4. ALGORITMO DE KRUSKAL

- Entrada : Grafo conexo $G = (V, E)$ y vector de costos $c \in \mathbb{R}^E$

- Salida : $H = (V, T)$ árbol generador de costo mínimo para G .

- $T := \emptyset$

- Ordenar las aristas de E de acuerdo a sus costos:

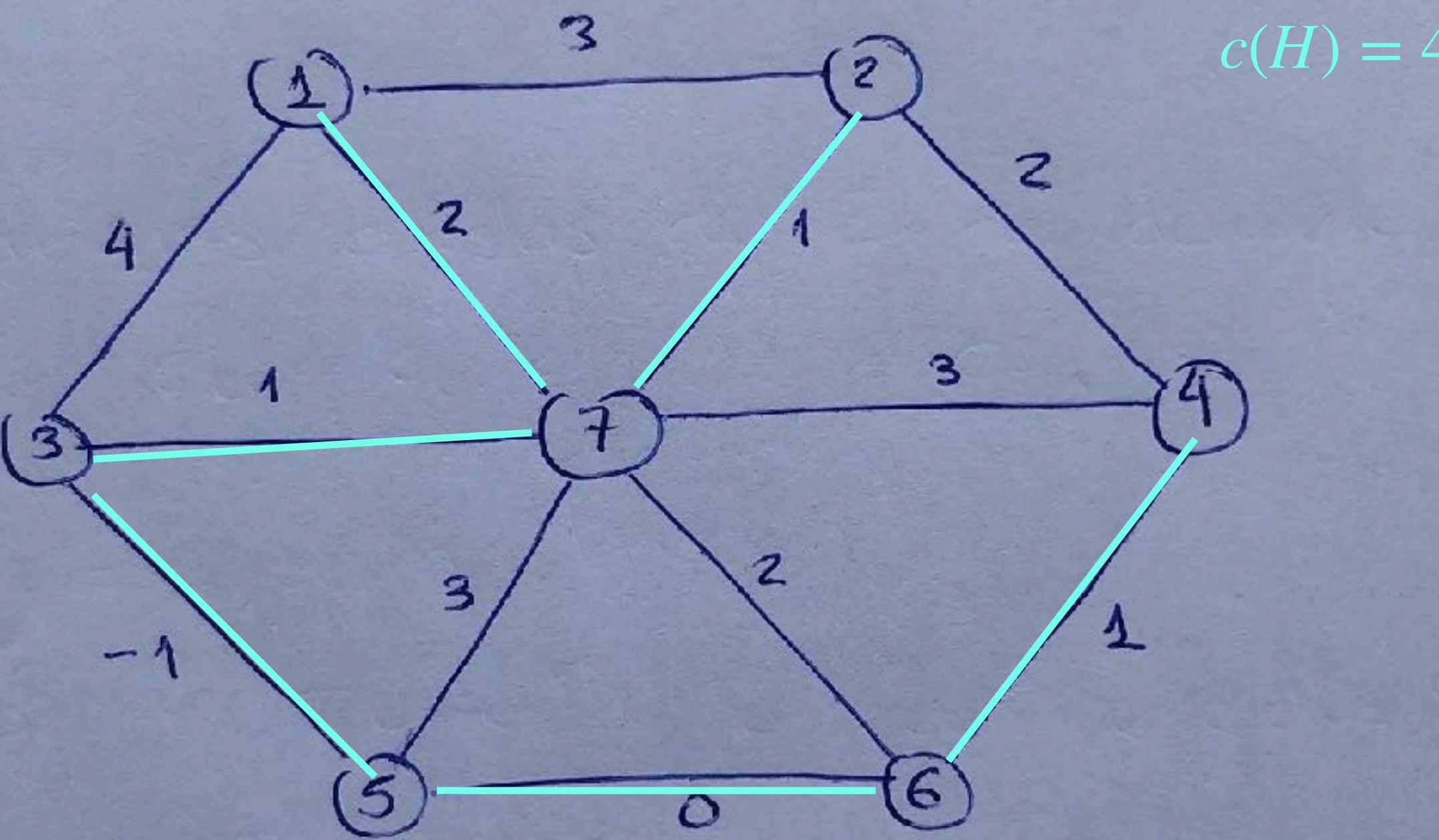
$$c_{e_1} \leq c_{e_2} \leq \dots \leq c_{e_m}$$

- Para $i = 1, \dots, m$ hacer :

- Si $(V, T \cup \{e_i\})$ no contiene ciclos:
 - + $T := T \cup \{e_i\}$

Ejercicio :

Encontrar un MST empleando el algoritmo de Kruskal



$$c_{35} \leq c_{56} \leq c_{46} \leq c_{37} \leq c_{27} \leq c_{24} \leq c_{17} \leq c_{67} \leq c_{57} \leq c_{47} \leq c_{12} \leq c_{13}$$

Valididad del algoritmo de Kruskal

Teorema 1.5. El algoritmo de Kruskal termina en un número finito de pasos y retorna un árbol generador de costo mínimo para G .

Demonstración:

- El algoritmo termina en un número finito de pasos, pues consiste de un lazo principal que se ejecuta m veces.
- Para demostrar que encuentra un árbol generador de peso mínimo, vamos a notar por $T^{(i)}$ al valor de T luego de la i -ésima iteración del lazo principal, con $T^{(0)} = \emptyset$ el valor inicial.

- Notar que, por definición del algoritmo, $(V, T^{(i)})$ no contiene ciclos para $i \in \{0, \dots, m\}$. Además,
- $$T^{(0)} \subset T^{(1)} \subset T^{(2)} \subset \dots \subset T^{(m)}$$
- En particular, $(V, T^{(m)})$ no contiene ciclos. Vamos a demostrar que este grafo es conexo. En efecto, asumamos que no lo es. Esto significa que tiene al menos dos componentes conexas, es decir, que existe SCV t.q.

$\check{\delta(S)} \cap T^{(m)} = \emptyset$

$$\delta(S) := \{kj \in E : k \in S \wedge j \notin S\}$$

Pero como G es conexo, sabemos que $\delta(S) \neq \emptyset$
y luego existe $e_i \in \delta(S) \setminus T^{(m)}$.

Supongamos que $e_i = jk$, con $j \in S$, $k \notin S$. En la i -ésima iteración del algoritmo se consideró la posibilidad de añadir e_i a $T^{(i-1)}$ para formar $T^{(i)}$. Como esto no ocurrió, significa que $T^{(i-1)} \cup \{e_i\}$ forma un ciclo.

Pero entonces, como $T^{(i-1)} \subset T^{(m)}$, se tiene que $T^{(m)} \cup \{e_i\}$ forma un ciclo. Es decir, $(V, T^{(m)})$ contiene un sendero desde $j \in S$ hasta $k \notin S$. Esto implica que $T^{(m)}$ contiene una amistad de $\delta(S)$ ↗ (contradicción). (Por qué?)

$\Rightarrow (V, T^{(m)})$ es conexo y sin ciclos

$\Rightarrow (V, T^{(m)})$ es un árbol generador de G .

- Para demostrar que $(V, T^{(m)})$ es de costo mínimo, probaremos por inducción que $T^{(i)}$ es un conjunto extensible, para $i \in \{0, \dots, m\}$.

$T^{(0)} = \emptyset$ es extensible.

Sea $i \in \{1, \dots, m\}$. Supongamos que $T^{(i-1)}$ es extensible y probemos que $T^{(i)}$ es extensible.

Según la definición del algoritmo, hay dos casos posibles:

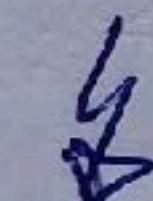
(i) $T^{(i)} = T^{(i-1)}$: Por hipótesis de inducción se tiene que $T^{(i)}$ es extensible.

(ii) $T^{(i)} = T^{(i-1)} \cup \{e_i\}$: En este caso $(V, T^{(i-1)} \cup \{e_i\})$

no contiene ciclos. Es decir, si $e_i = jk$, entonces j y k pertenecen a componentes conexas distintas del grafo $(V, T^{(i-1)})$. Sea $S \subset V$ la componente conexa que contiene a j . Tenemos que $e_i \in \delta(S)$. Y además $\delta(S) \cap T^{(i-1)} = \emptyset$.

Supongamos ahora que existe una arista $e_l \in \delta(S)$ con $l < i$. Como las aristas están ordenadas por sus costos, esto implica que $l < i$. Pero entonces, en ^{una} ~~la~~ iteración previa se consideró añadir e_l a $T^{(l-1)}$.

Notar que $e_l \notin T^{(i-1)}$ (Por qué?). Es decir, e_l no

fue añadida a $T^{(l-1)}$. De acuerdo a la definición del algoritmo, esto implica que $T^{(l-1)} \cup \{e\}$ contiene un ciclo. Pero entonces, como $T^{(l-1)} \subset T^{(i-1)}$ se sigue que $T^{(l-1)} \cup \{e\}$ contiene un ciclo, lo que significa que $(V, T^{(i-1)})$ contiene un sendero que conecta los dos extremos de e . Como $e \in \delta(S)$, este sendero contiene al menos una arista de $\delta(S)$.
(Ejercicio) 

Es decir, e es una arista de costo mínimo de $\delta(S)$. Aplicando el Lemma 1.3., concluimos que $T^{(i)}$ es extensible.

$\Rightarrow T^{(m)}$ es extensible

$\Rightarrow (V, T^{(m)})$ es un árbol generador de peso mínimo.

Eficiencia del algoritmo de Kruskal

- $T = \emptyset$ $O(1)$
 - Ordenar aristas de E : $c_{e_1} \leq c_{e_2} \leq \dots \leq c_{e_m}$ $O(m \cdot \log m)$
 - Para $i = 1, \dots, m$, hacer:
 - Si $(V, T \cup \{e_i\})$ no contiene ciclos: ?
 - $T := T \cup \{e_i\}$ $O(1)$
- m-veces

Para determinar si se forma o no un ciclo al agregar la arista e_i en el lazo principal, puede utilizarse un vector $S \in \mathbb{Z}^V$ que almacene para cada nodo $j \in V$ la componente conexa $S[j]$ del grado (V, T) a la que pertenece j . Con esto, el algoritmo puede implementarse de la siguiente manera:

Iniciar:

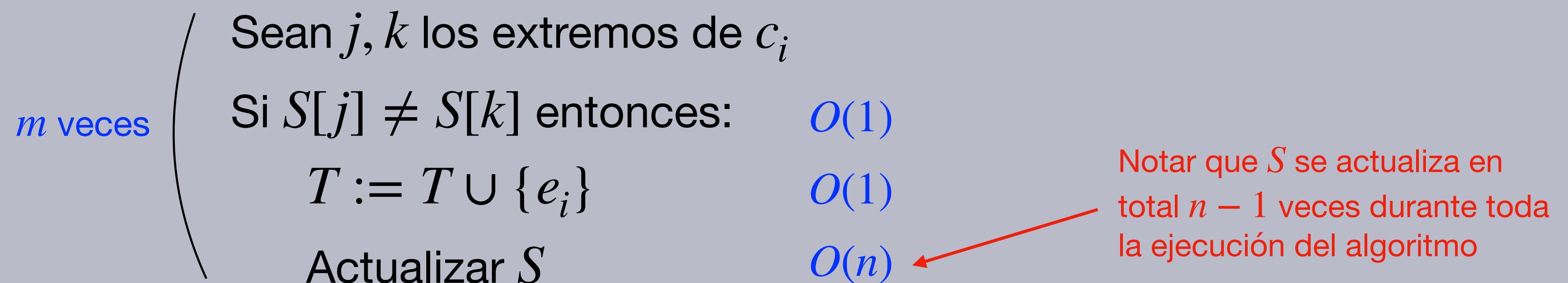
$$T := \emptyset; \quad O(1)$$

$$S[j] := j, \forall j \in V \quad O(n)$$

Ordenar las aristas según su costo: $O(m \log m)$

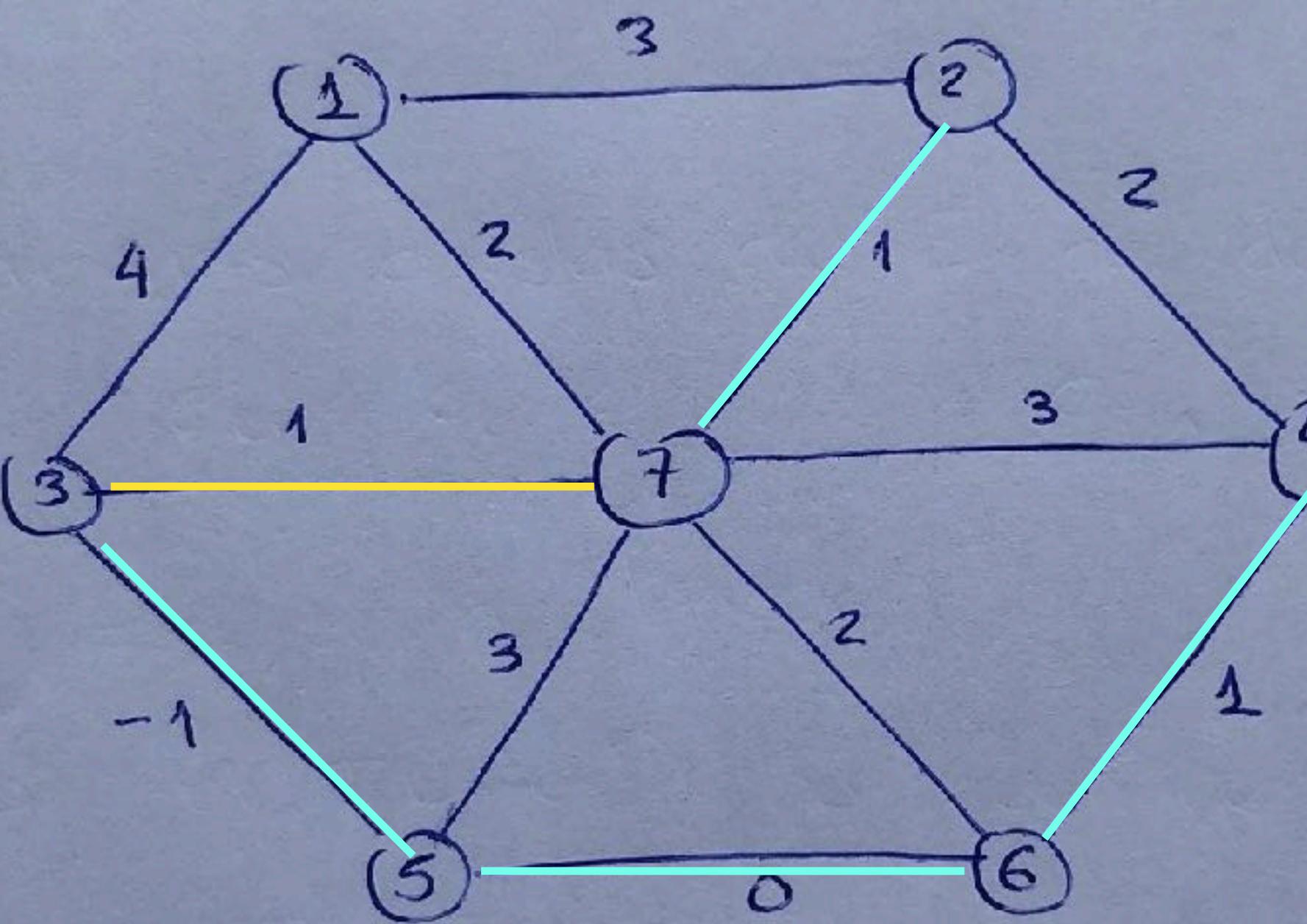
$$c_1 \leq c_2 \leq \dots \leq c_m$$

Para $i := 1, \dots, m$ hacer:



Ejercicio :

Encontrar un MST empleando el algoritmo de Kruskal



$$c_{35} \leq c_{56} \leq c_{46} \leq c_{27} \leq c_{37} \leq c_{24} \leq c_{17} \leq c_{67} \leq c_{57} \leq c_{47} \leq c_{12} \leq c_{13}$$

S:

1	2	3	4	5	6	7
1	2	3	3	3	3	2

La actualización de S consiste en recorrer el vector y sustituir el valor de $S[j]$ por $S[k]$, o viceversa. Notar que esto requiere $O(n)$ operaciones y se realiza en total $n - 1$ veces. Por lo tanto, el algoritmo de Kruskal requiere:

$$O(n) + O(m \log m) + O(m) + O(n^2) = O(m \log m) + O(n^2) \text{ operaciones}$$

La actualización de S puede hacerse de manera más eficiente si se almacenan listas con los nodos que pertenecen a cada componente conexa. En este caso, puede conseguirse actualizar S en $O(\log n)$ operaciones, y el algoritmo de Kruskal requiere en total:

$$O(m \log m) + O(n \log n) = O(m \log m) \text{ operaciones}$$

1.5. MATROIDES Y EL ALGORITMO GLOTÓN

El algoritmo de Kruskal es un ejemplo de un algoritmo glotón o algoritmo codicioso [greedy algorithm]:

En cada iteración, toma localmente la mejor decisión.

Generalmente, los algoritmos-glotones no producen soluciones óptimas en los problemas de optimización combinatoria.

Un caso importante donde sí lo hacen, que incluye al MST como un caso particular, es en la optimización sobre matroides.

Definición. (Sistemas de independencia)

Sean X : un conjunto finito

\mathcal{I} : una familia de subconjuntos de X , $\mathcal{I} \subseteq 2^X$

El par (X, \mathcal{I}) es un sistema de independencia [independence system] si se cumplen los axiomas:

(I1) : $\emptyset \in \mathcal{I}$

(I2) Si $A \in \mathcal{I}$ y $B \subseteq A \Rightarrow B \in \mathcal{I}$ (prop. hereditaria)

Los elementos de \mathcal{I} se llaman conjuntos independientes.

Ejemplos:

1) Sea $G = (V, E)$ un grafo no dirigido.

Si $X = E$, y

$\mathcal{I} = \{F \subseteq E : F \text{ es el conjunto de aristas de un bosque en } G\}$

Entonces (X, \mathcal{I}) es un sistema de independencia.

2) Sean $G = (V, E)$, $X = E$ y

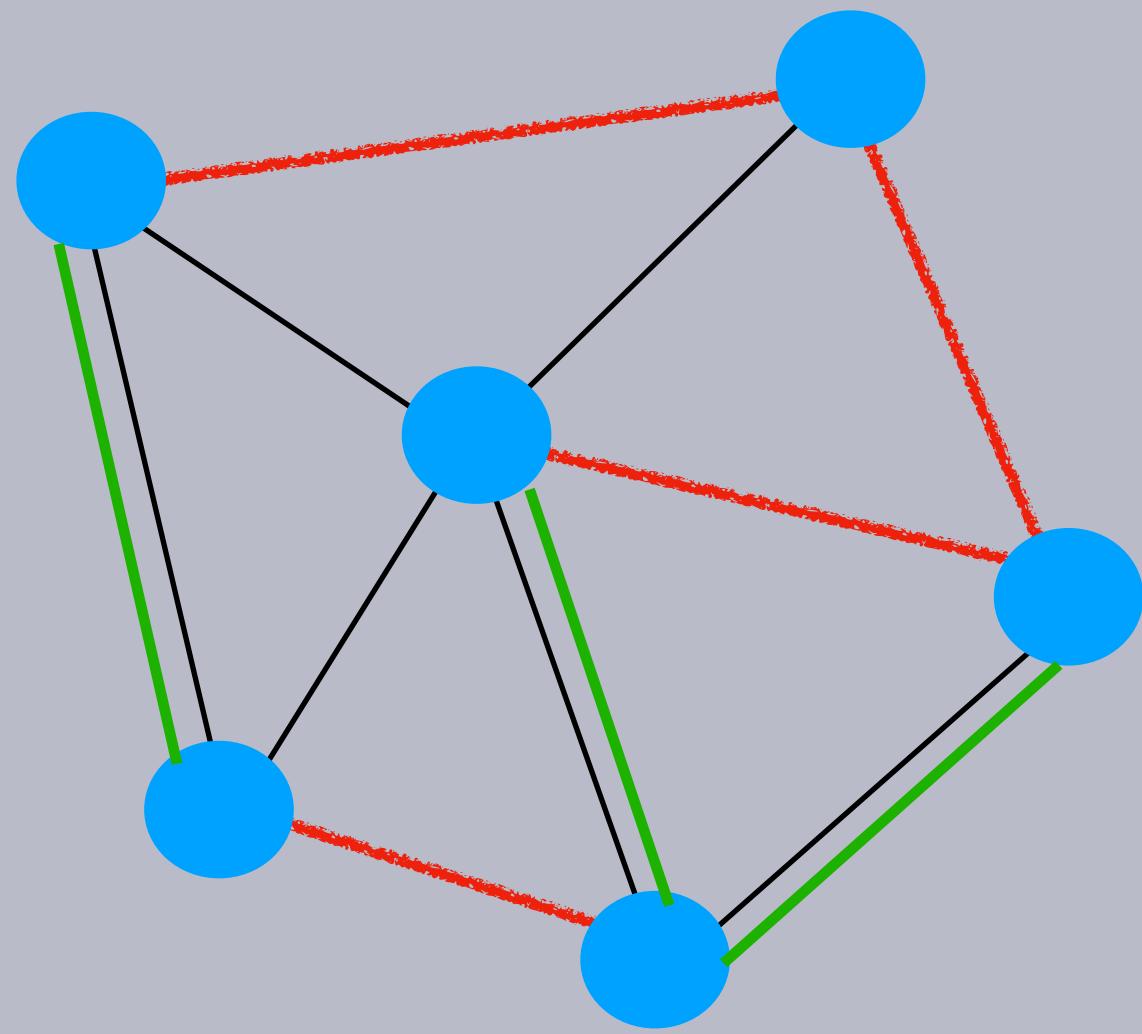
$\mathcal{I} = \{F \subseteq E : \text{ningún par de aristas de } F \text{ son incidentes a un mismo nodo}\}$

3) Sean $G = (V, E)$, $X = V$ y

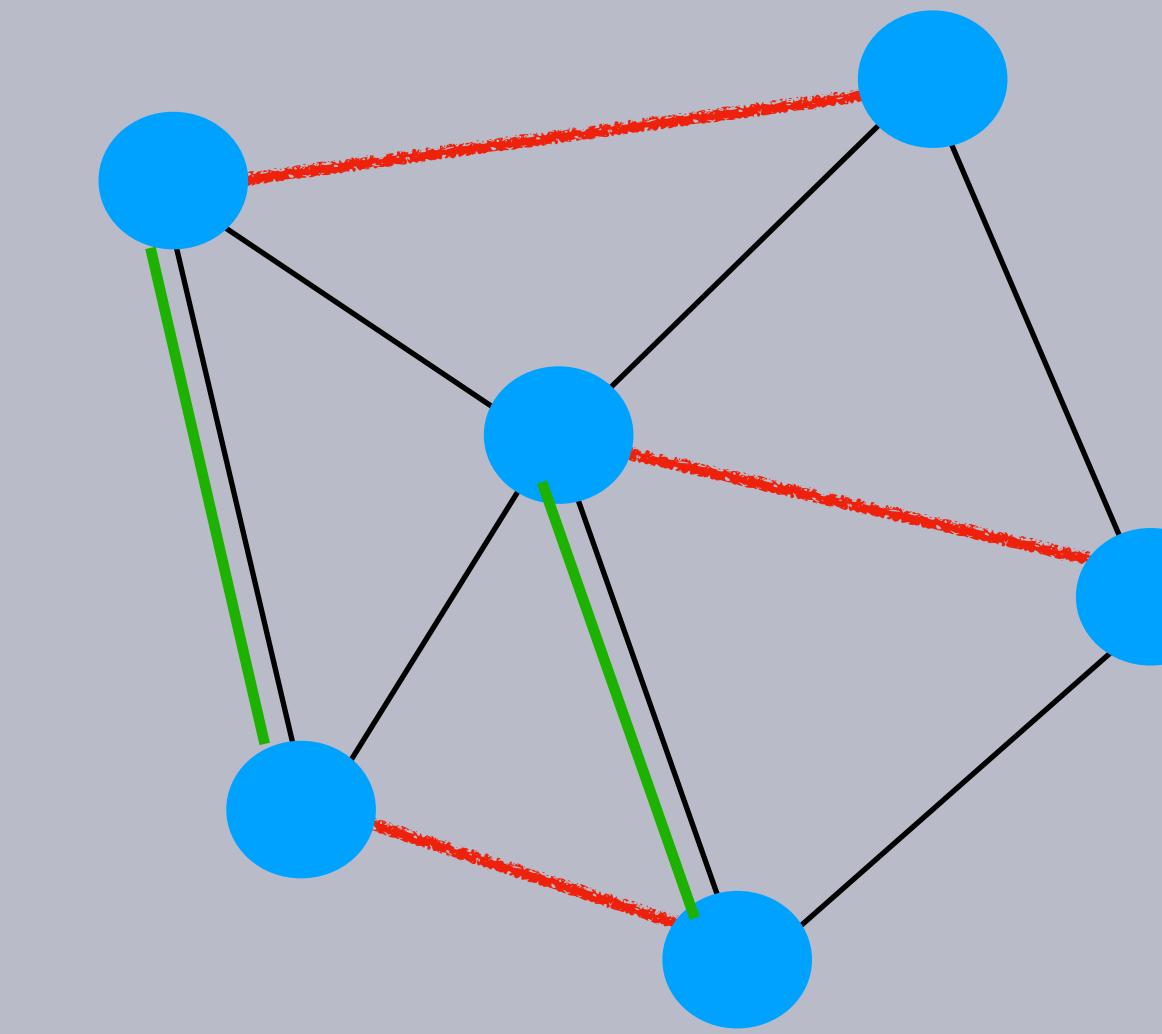
$\mathcal{I} = \{W \subseteq V : \text{ningún par de nodos en } W \text{ son adyacentes}\}$
(conjuntos estables)

4) Sean $X \subseteq \mathbb{R}^n$, finito, y $\mathcal{I} = \{V \subseteq X : V \text{ es l.i.}\}$.

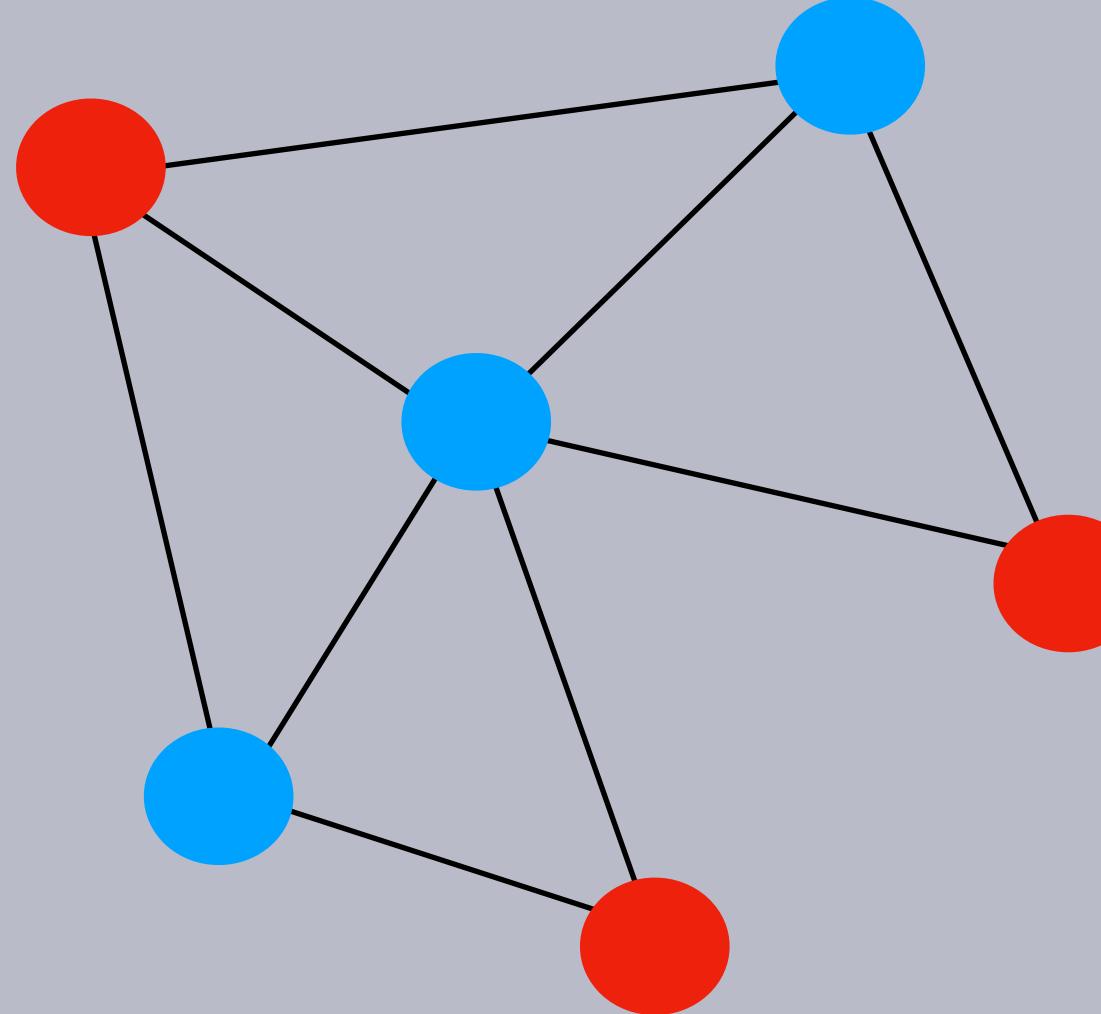
Ejemplo 1



Ejemplo 2



Ejemplo 3



Ejemplo 4

$$X = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\}$$

$$V = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

Notación:

- Los subconjuntos de X que pertenecen a \mathcal{I} se llaman conjuntos independientes, los demás se llaman conjuntos dependientes. [Ejemplos]
- Los conjuntos independientes maximales se llaman bases. [Ejemplos]
- Los conjuntos dependientes minimales se llaman circuitos. [Ejemplos]

Definición: (Matroide)

Un sistema de independencia (X, \mathcal{I}) es un matroide, si cumple

(I3) Si $A, B \in \mathcal{I}$ con $|A| < |B|$, entonces existe $x \in B \setminus A$ t.q. $A \cup \{x\} \in \mathcal{I}$. (prop. de aumentación) [Ejemplos]

Propiedades de los matroides

Sea (X, \mathcal{I}) un matroide.

(i) Si A, B son dos bases, entonces $|A|=|B|$.

(ii) Si A, B son bases distintas, $a \in A \setminus B$ y $b \in B \setminus A$, entonces $(A \cup \{b\}) \setminus \{a\}$ es una base.

(Propiedad de intercambio de bases) [Ejercicio].

(iii) Si $F \subseteq X$, $A \subseteq F$, $B \subseteq F$ \wedge A, B son independientes $\Rightarrow |A|=|B|$
maximales en F .

Dem:

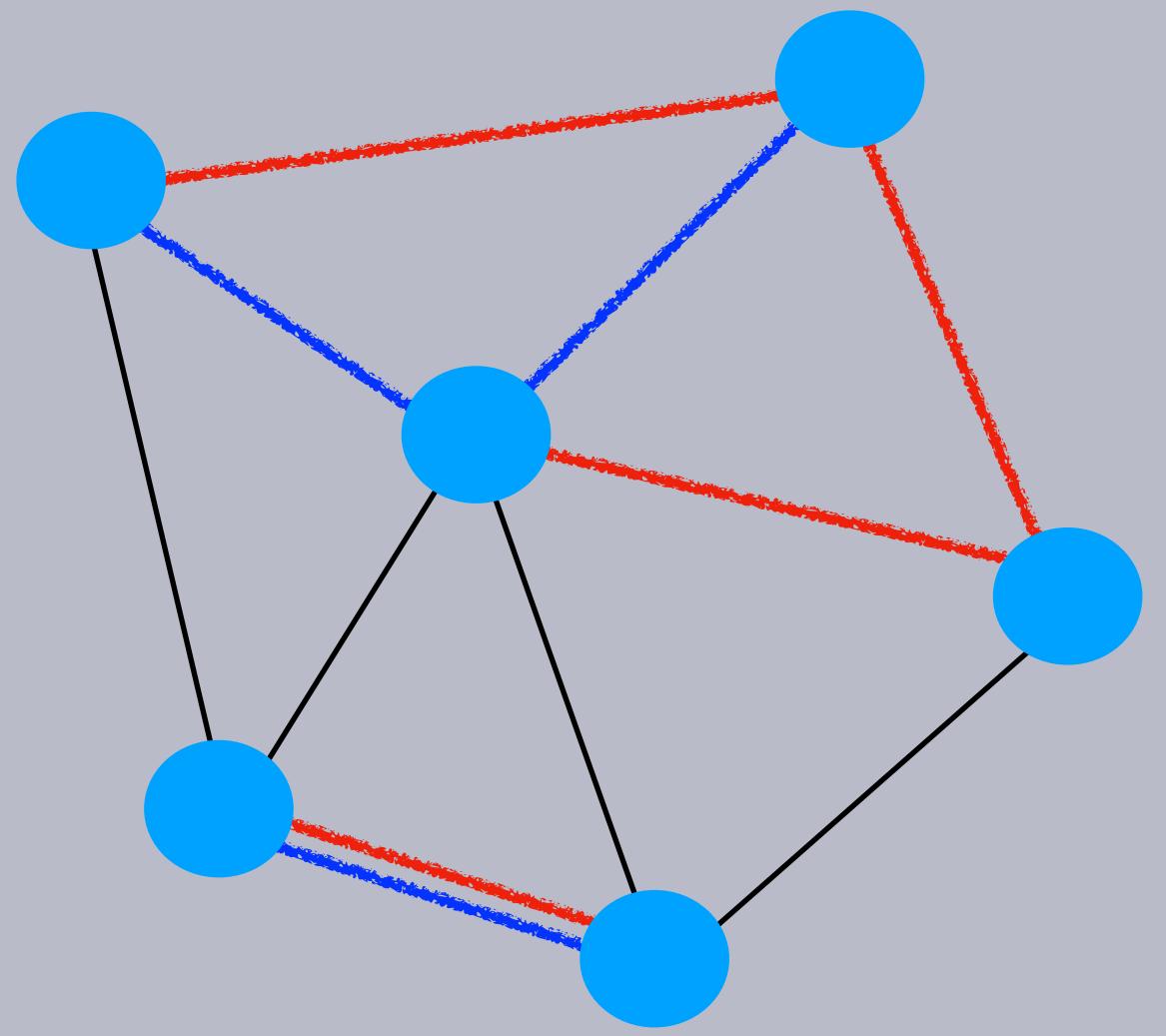
(i) Suponer que existen dos bases $|A|, |B|$, con $|A| < |B|$.

Entonces por (I3) existe $x \in B \setminus A$ t.g. $A \cup \{x\} \in \mathcal{I}$.

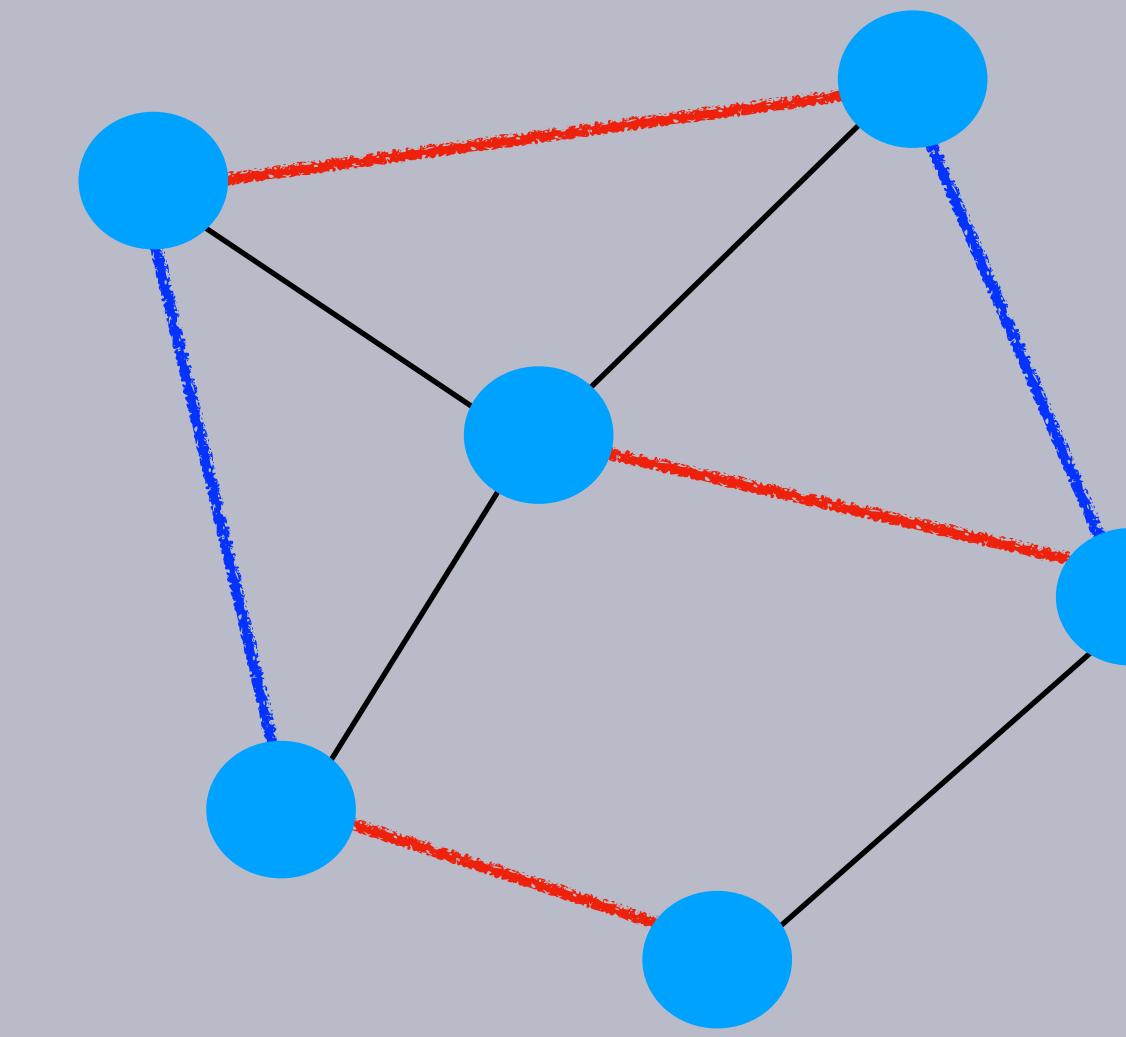
Luego, A no es maximal. ↴

(iii) Similar a (i)

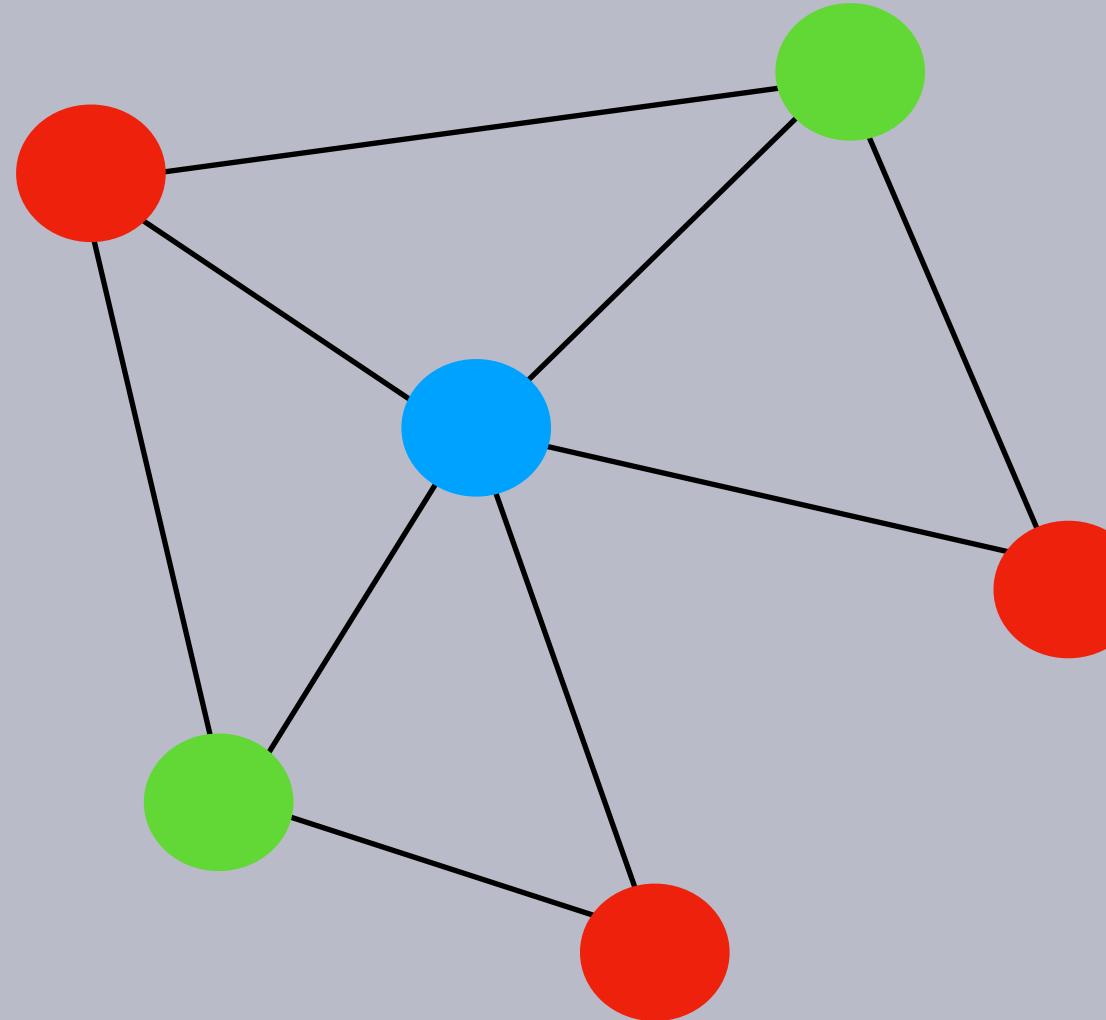
Ejemplo 1: Es un matroide



Ejemplo 2: No es un matroide



Ejemplo 3: No es un matroide



Ejemplo 4: Es un matroide

$$X = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\}$$

$$V_1 = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

$$V_2 = \left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \right\}$$

Problema (Base de peso máximo)

Sean (X, \mathcal{I}) un matroide, y $w \in \mathbb{R}^X$ un vector de pesos asociados a los elementos de X .

Para $A \subseteq X$, definimos $w(A) := \sum_{i \in A} w_i$.

Encontrar una base de peso máximo.

Algoritmo (Glotón)

- $B := \emptyset$
- Ordenar los elementos de X de manera descendente respecto a sus pesos
 $w_{i_1} \geq w_{i_2} \geq \dots \geq w_{i_{n-1}} \geq w_{i_n}$, con $n := |X|$
- Para $k := 1, \dots, n$
 - ↳ Si $B \cup \{i_k\} \in \mathcal{I}$, entonces $B := B \cup \{i_k\}$

Teorema 1.6.

El algoritmo glotón encuentra la solución óptima al problema de la base de peso máximo en un matroide.

Demonstración

- Notar que el algoritmo termina en un número finito de pasos y retorna una base B del matroide.
- Suponer que $B = \{b_1, \dots, b_m\}$ no es una base de peso máximo y sea $B' := \{b'_1, \dots, b'_m\}$ una base de peso estictamente mayor a B . Asumimos que $w_{b_1} \geq w_{b_2} \geq \dots \geq w_{b_m}$ y $w_{b'_1} \geq w_{b'_2} \geq \dots \geq w_{b'_m}$

Sea ℓ el menor índice tal que $w_{b_\ell} > w_{b_l}$.
(Notar que tal índice debe existir. Por qué?)

Sea

$$F := \{b_1, \dots, b_{\ell-1}\} \cup \{b'_1, \dots, b'_{\ell-1}, b'_\ell\}$$

Notar que $w_{b'_1} \geq w_{b'_2} \geq \dots \geq w_{b'_{\ell-1}} \geq w_{b'_\ell} > w_{b_\ell}$.

Por lo tanto, en una iteración anterior a la iteración en la que se agregó el elemento b_ℓ a B , el algoritmo consideró agregar cada uno de los elementos b'_i , con $i=1, \dots, \ell$.

Si el algoritmo lo hizo, entonces $b'_i \in \{b_1, \dots, b_{\ell-1}\}$

Caso contrario, $\{b_1, \dots, b_{\ell-1}\} \cup \{b'_i\} \notin \mathcal{I}$

En cualquier caso, se concluye que $\{b_1, \dots, b_{l-1}\}$ es un subconjunto independiente maximal de F .

Por otra parte, $\{b'_1, \dots, b'_{l-1}, b_l\}$ es un subconjunto independiente de F y además

$$|\{b'_1, \dots, b'_l\}| = l > l-1 = |\{b_1, \dots, b_{l-1}\}| \quad \text{(Propiedad (iii))}$$

— o —

El algoritmo glotón puede aplicarse a sistemas de independencia en general, pero no necesariamente producirá una solución óptima.

Teorema 1.7.

Sea (X, I) un sistema de independencia

(X, I) es un matroide

\Leftrightarrow el algoritmo glotón encuentra una base de peso máximo para todo $w \in \mathbb{R}^X$.