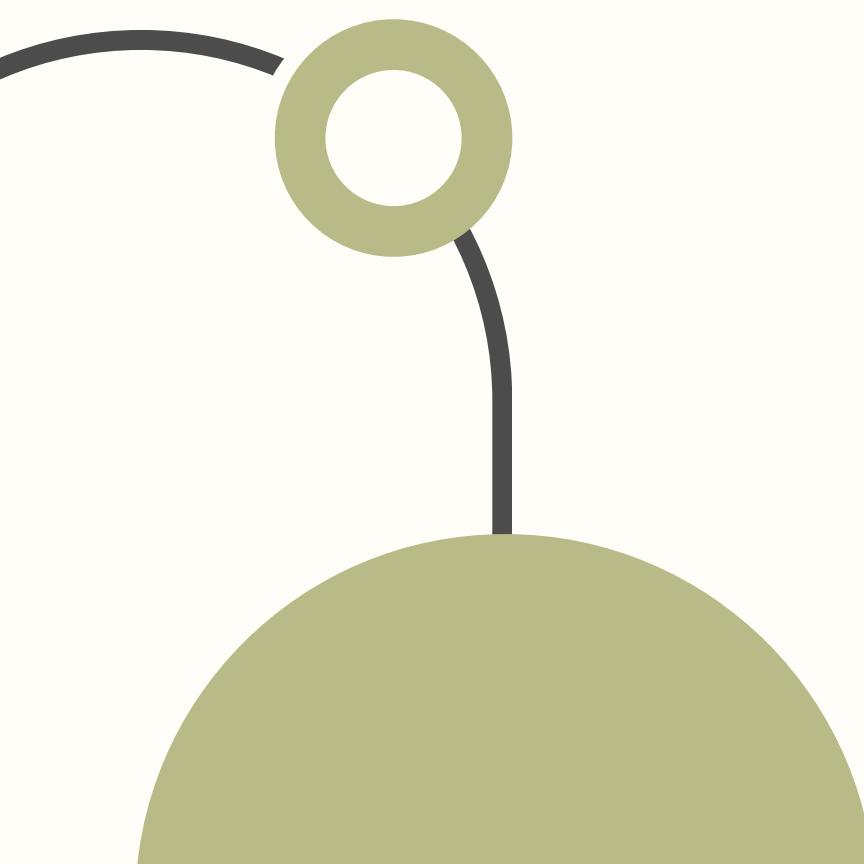


GRAFOS III



Realizado por:

Luis Ángel Peña Mungarro
Edgar Gerardo Vargas Espinosa
Julio César Puente Reynoso
Diego Granja Peña

Índice de CONTENIDOS



-
- 1. Grafos no dirigidos,
definición, conceptos.**

 - 2. Representación de grafos
no dirigidos.**

 - 3. Recorridos de un grafo
(Anchura y Profundidad).**

 - 4. Árbol de expansión mínima
en grafos ponderados**

 - 5. Métodos de búsqueda**
-

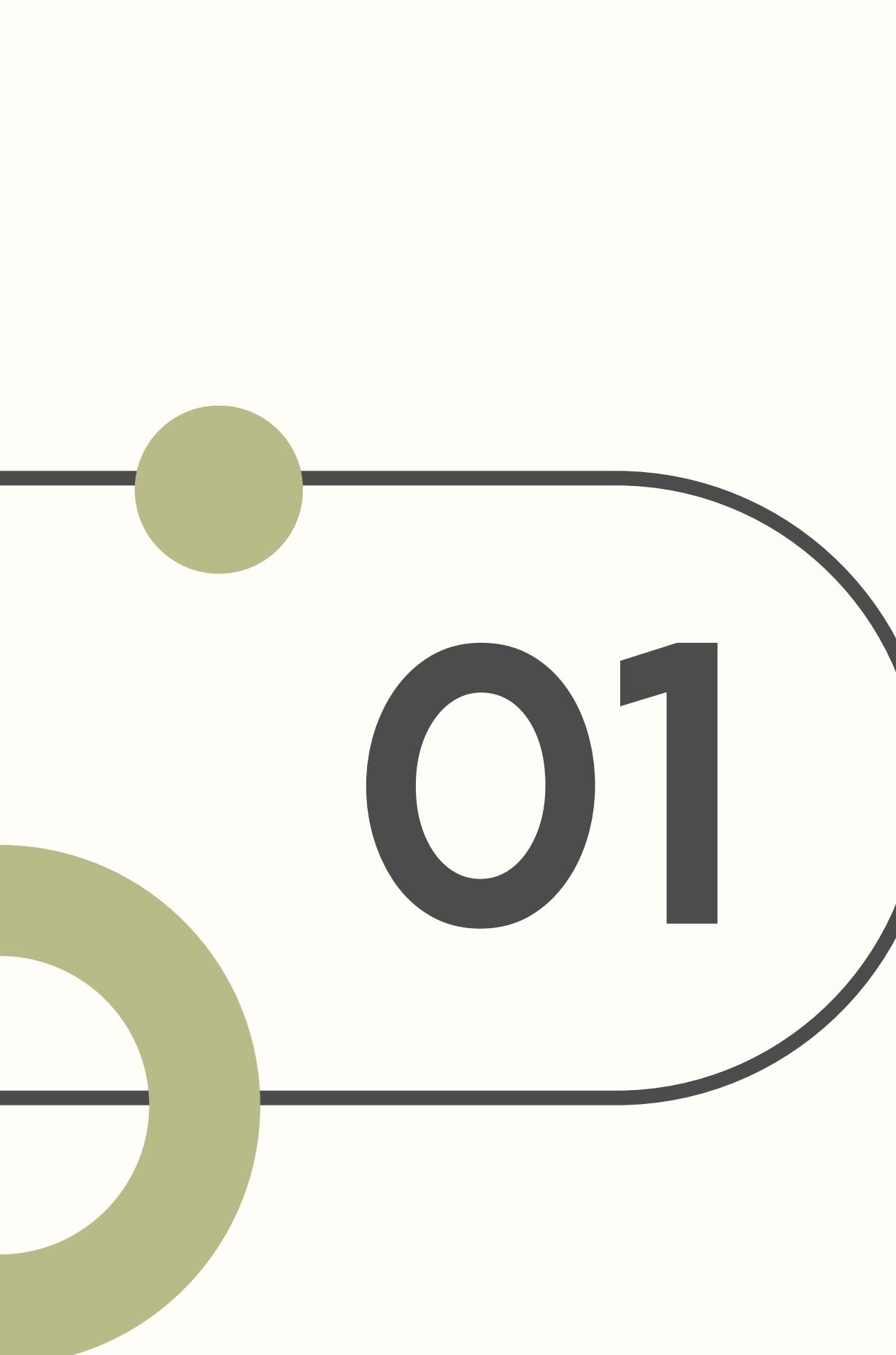
Índice de CONTENIDOS



6. Aplicaciones

7. Conclusión

8. Bibliografía

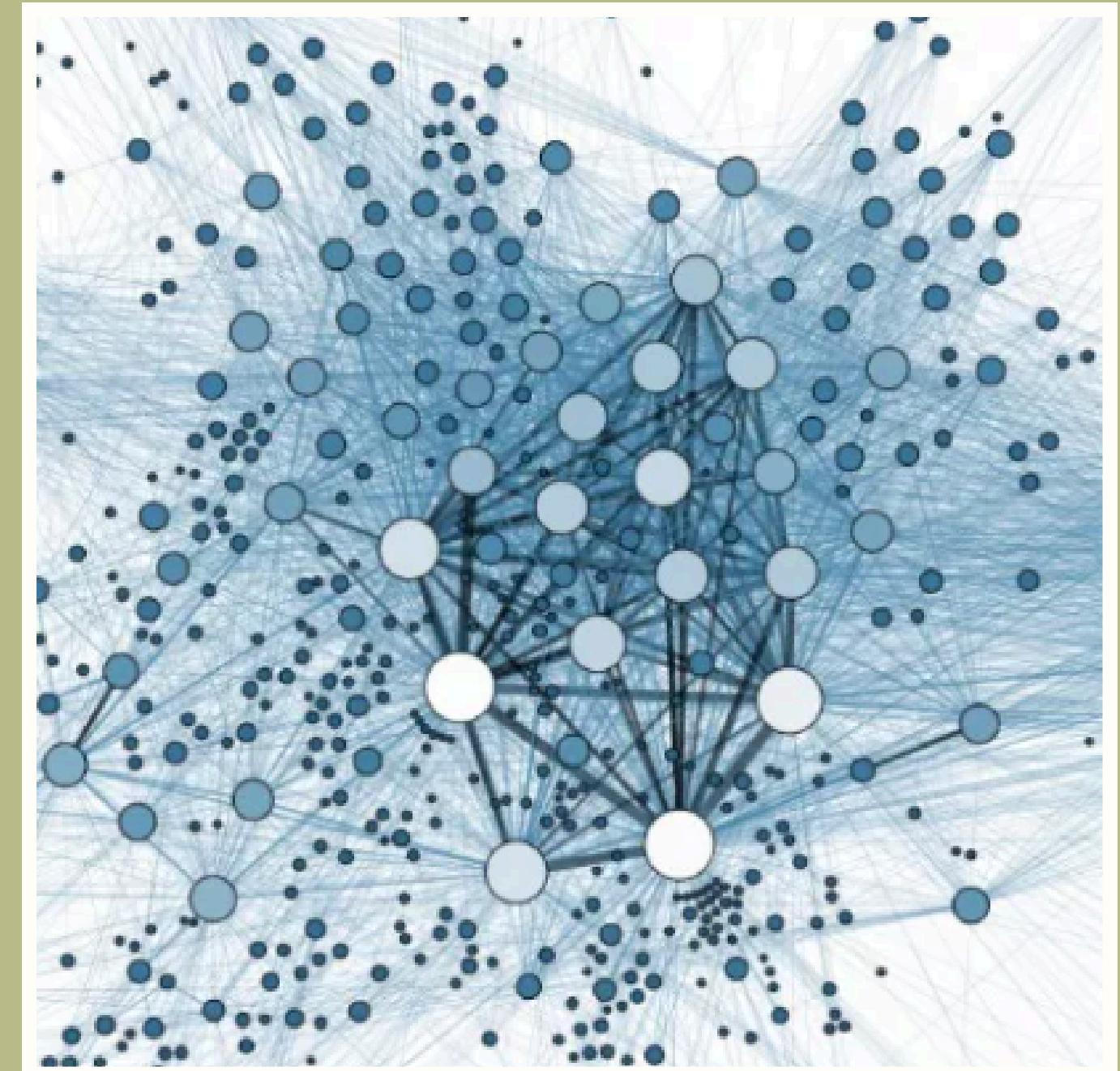


01

Grafos no
dirigidos

→ GRAFOS NO DIRIGIDOS

Un grafo no dirigido es una estructura matemática que consta de un conjunto de vértices (o nodos) y un conjunto de aristas (o enlaces) que conectan pares de vértices.





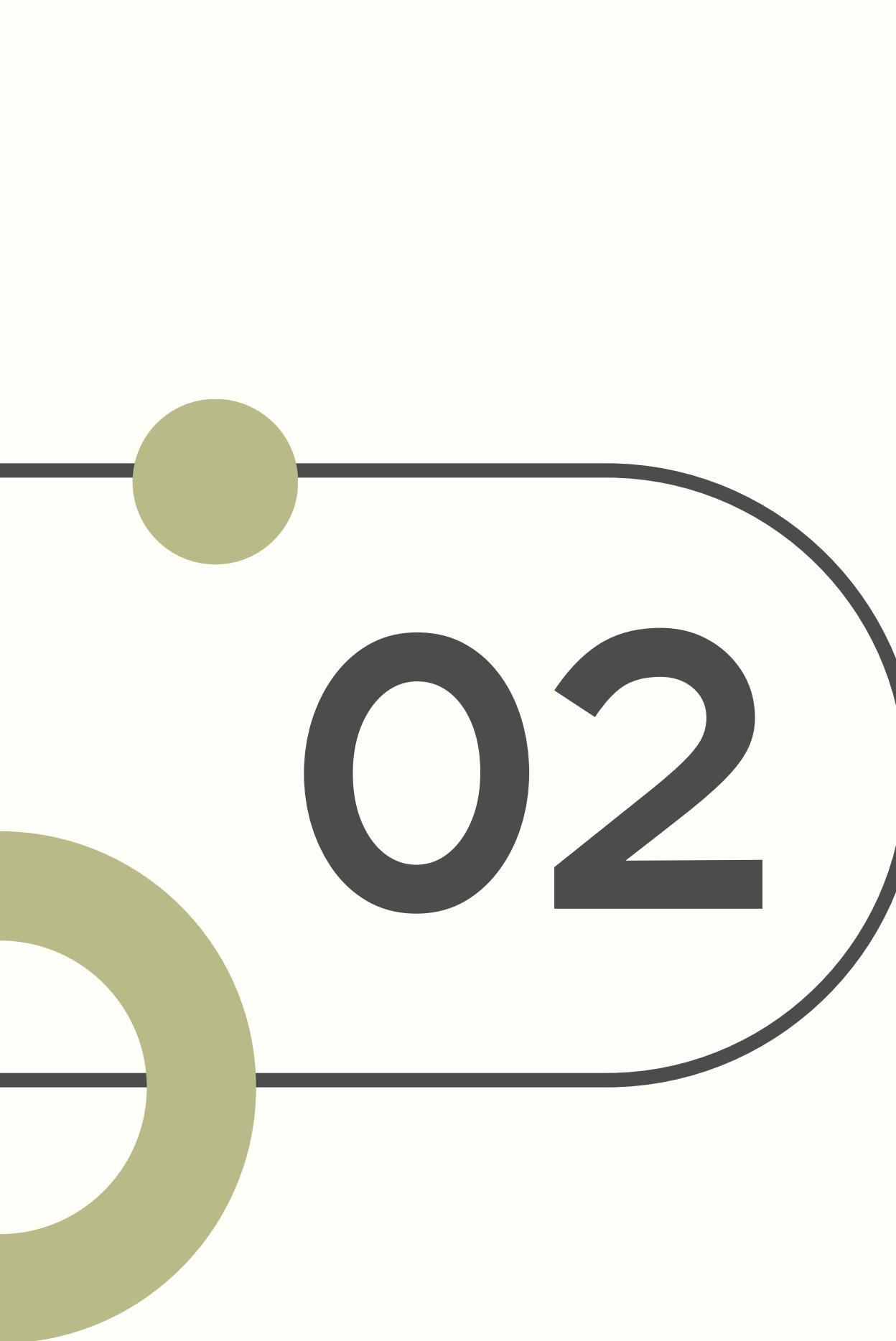
GRAFOS DIRIGIDOS

Un grafo dirigido está compuesto por vértices y aristas. Estas aristas tienen un solo sentido y por ende solo se puede recorrer en una sola dirección.

APLICACIONES

Representar grupos de amistades en redes sociales, representar lugares físicos como puntos importantes de ciudades, representar laberintos, relaciones entre objetos, etc.





02

Representación de grafos no dirigidos



REPRESENTACIÓN DE GRAFOS NO DIRIGIDOS.

- 01 Matriz de adyacencia
- 02 Lista de adyacencia
- 03 Matriz de incidencia

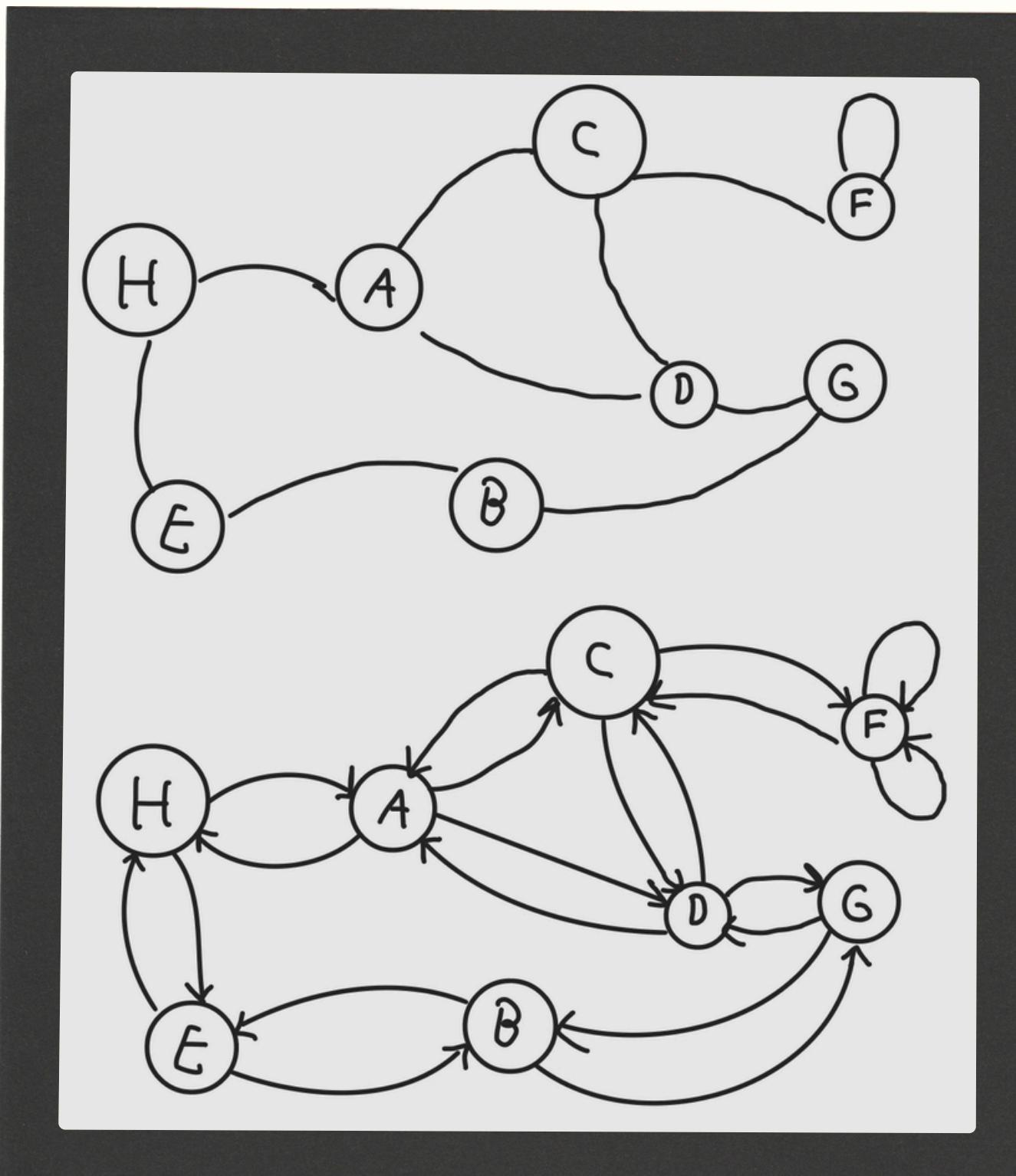
→ Grafo no dirigido como dirigido

Se puede representar un grafo no dirigido con un grafo dirigido.

Esto se hace generando 2 enlaces en el grafo no dirigido por cada enlace del grafo dirigido, uno de ida y otro de vuelta.

Nota:

- Esto también aplica cuando un nodo se conecta a si mismo
- Se debe considerar si se quiere tener ciclos o no, así también si las aristas tienen peso o no



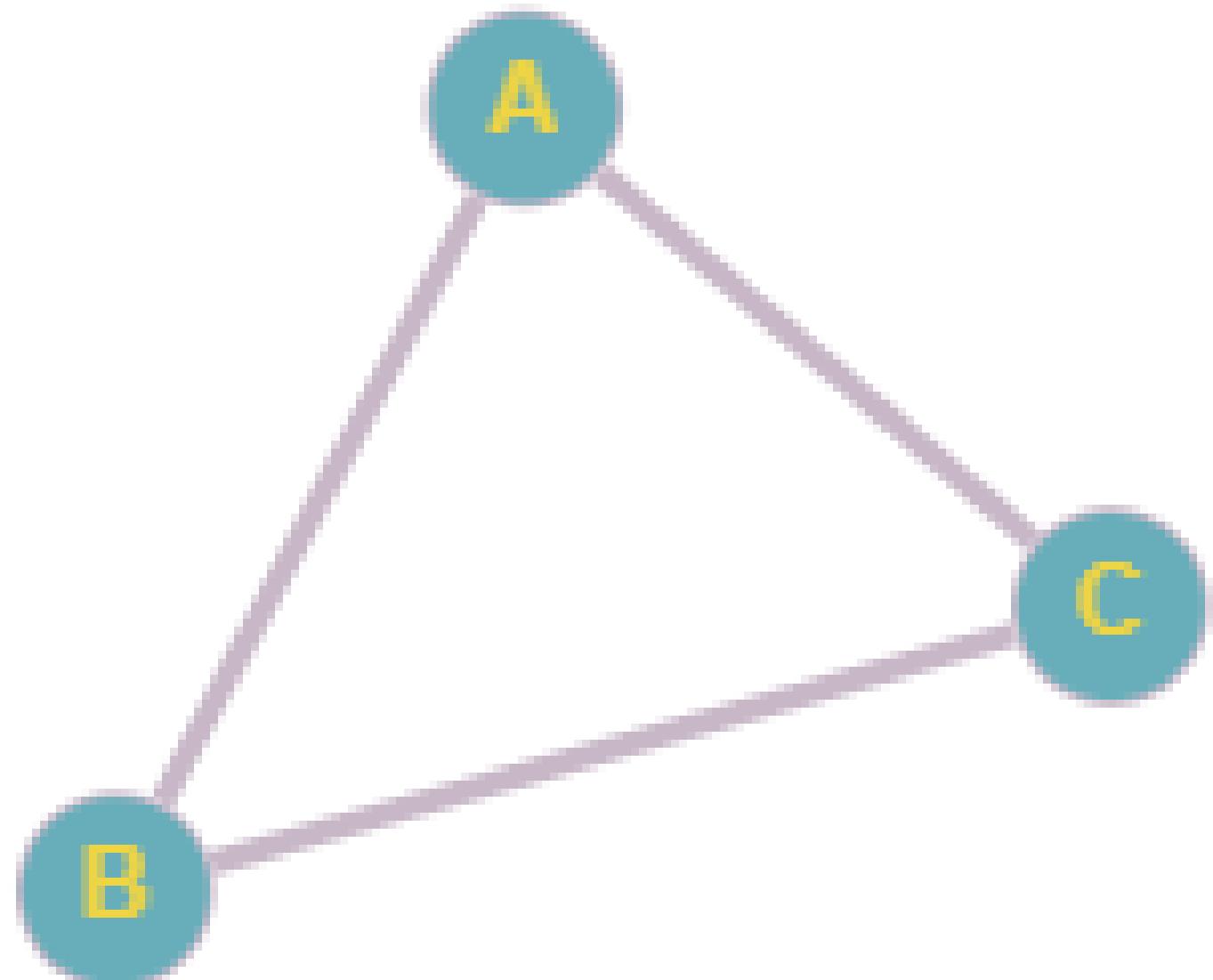
Ejemplo

- El nodo A está conectado al nodo C y B
- El nodo B está conectado al nodo A y C
- El nodo C está conectado al nodo A y B

$G(V, A)$

$V:\{A, B, C\}$

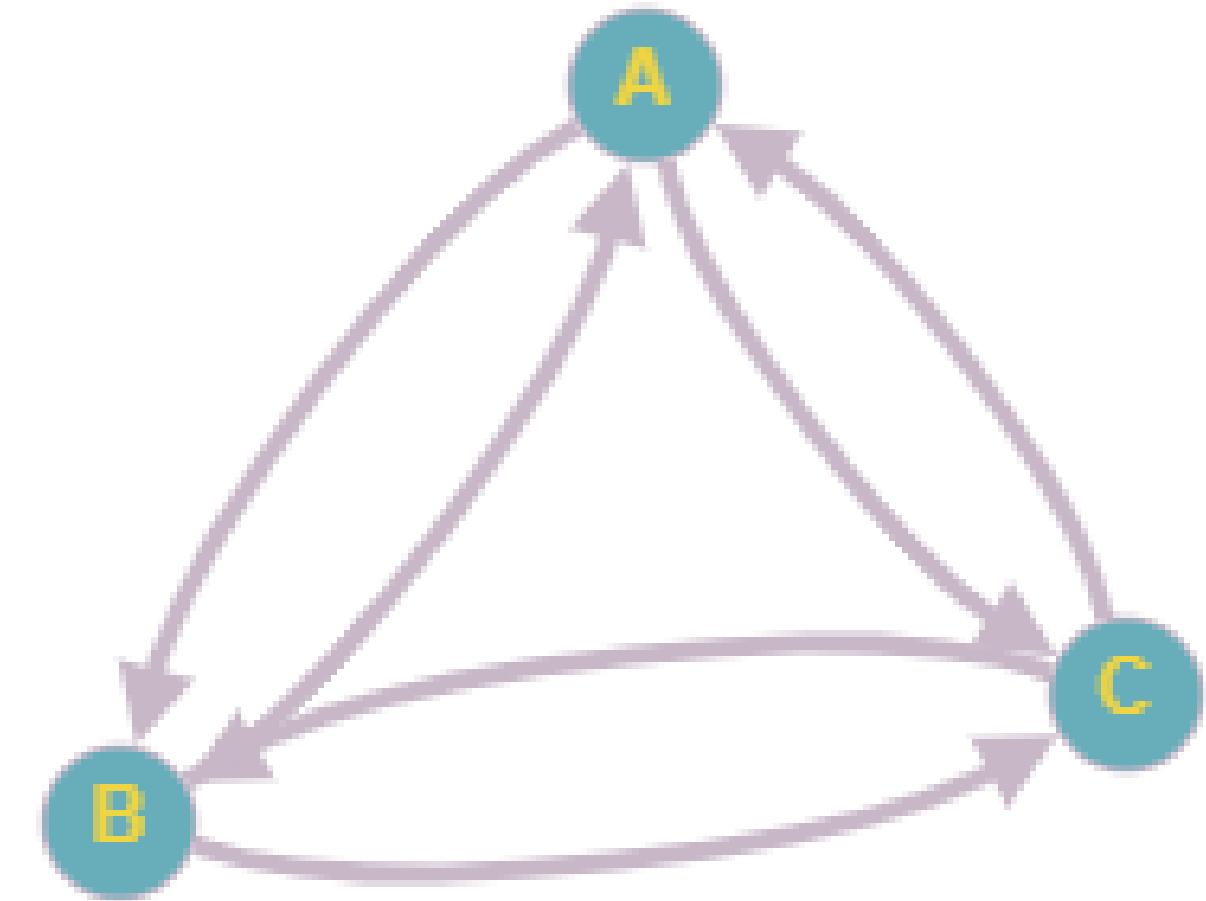
$A:\{(A,B), (B,A), (A,C), (C,A), (B,C), (C,B)\}$



Ejemplo

- Hacemos que el nodo A se diriga al nodo B y el nodo B al nodo A
- Hacemos que el nodo A se diriga al nodo C y el nodo C al nodo A
- Hacemos que el nodo B se dirige al nodo C y el nodo C al nodo B

El nodo A se dirige al nodo C y B
El nodo B se dirige al nodo A y C
El nodo C se dirige al nodo A y B



$G(V, A)$

$V:\{A, B, C\}$

$A:\{(A,B), (B,A), (A,C), (C,A), (B,C), (C,B)\}$

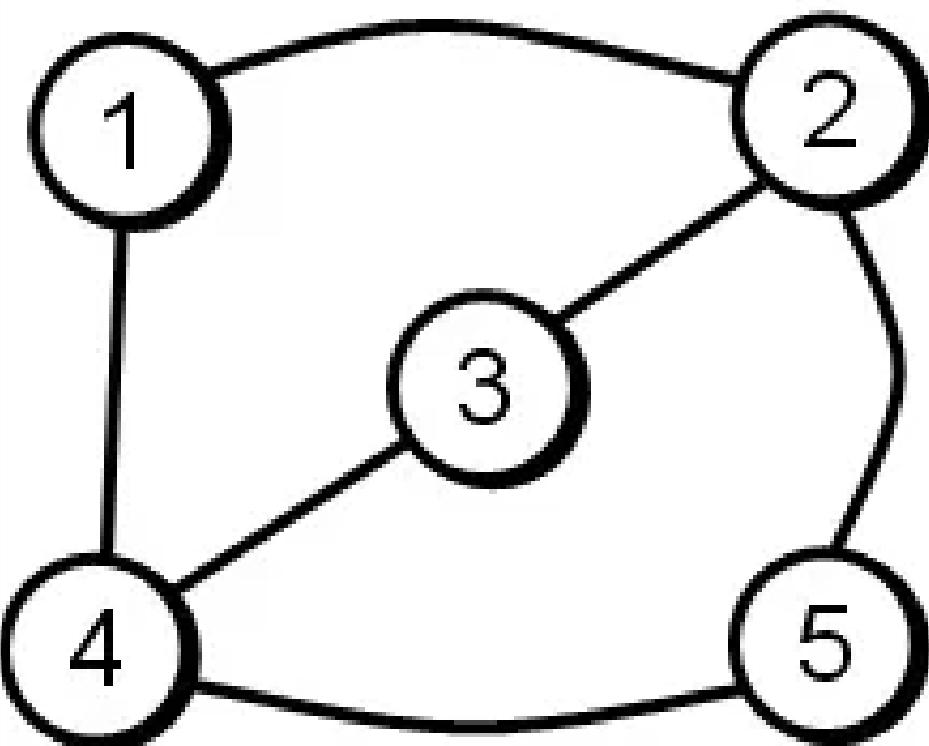
01 Matriz de adyacencia

Una matriz de adyacencia muestra los vértices de un grafo y la matriz muestra numéricamente cuantas aristas conectan ambos.

Notas:

- Si el grafo no es dirigido la matriz será simétrica, de ser un grafo dirigido no necesariamente será una matriz simétrica.
- Por lo general solo hay una sola arista conectando a cada nodo a menos de que conecte a sí misma, en ese caso habrá 2.

Útil para grafos densos
(muchas aristas)



$G(V, A)$

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

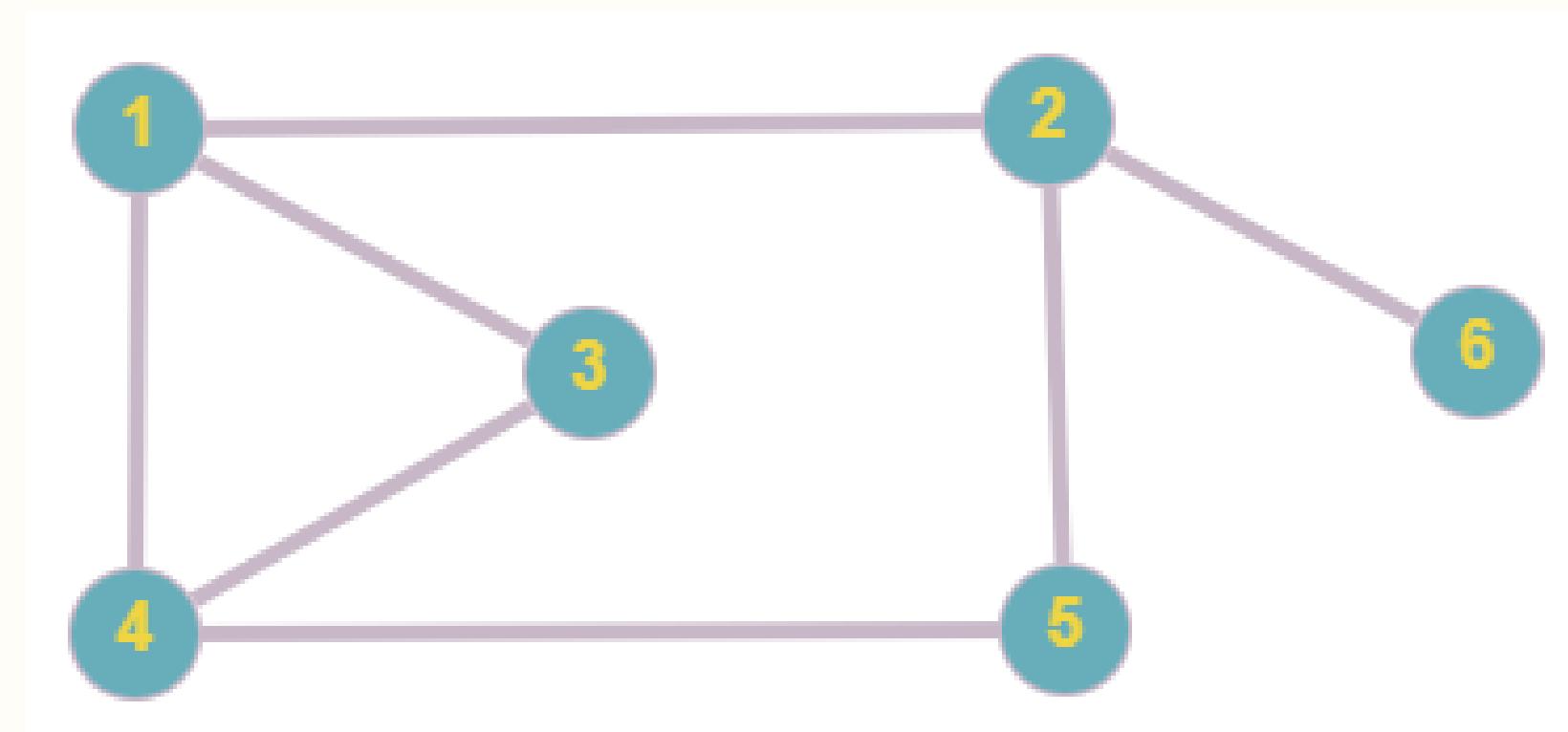
$A:\{(1,2), (2,3), (3,4), (4,5), (1,4), (2,5), (2,1), (3,2), (4,3), (5,4), (4,1), (5,2)\}$

$M(i,j) =$
0, Si no existe arista.
1, Si las aristas conectan.

M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

Ejemplo

Obtener la matriz de adyacencia del siguiente grafo:



G(V,A)

V:{ 1, 2, 3, 4, 5, 6}

A:{ (1,4), (2,5), (2,1), (3,1), (4,3), (5,4), (2,6)}

Ejemplo

M	1	2	3	4	5	6
1	0	1	1	1	0	0
2	1	0	0	0	1	1
3	1	0	0	1	0	0
4	1	0	1	0	1	0
5	0	1	0	1	0	0
6	0	1	0	0	0	0

G(V,A)

V:{ 1, 2, 3, 4, 5, 6}

A:{ (1,4), (2,5), (2,1), (3,1), (4,3), (5,4), (2,6)}

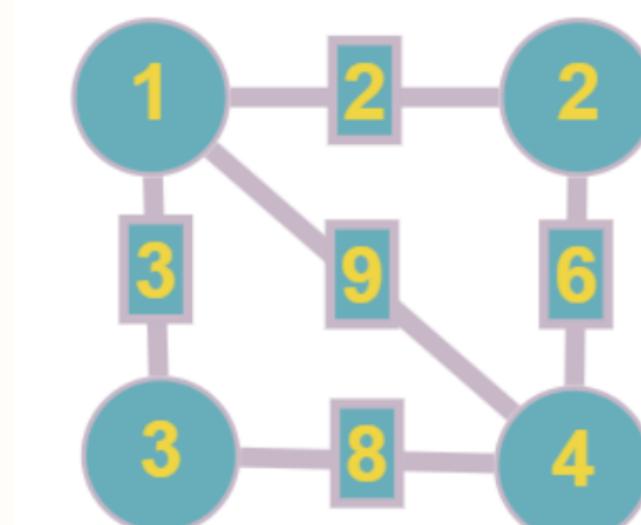
1.5

Matríz de adyacencia ponderada

Muestra un peso como valor numérico como arista entre los nodos.

Notas:

- Si el grafo no es dirigido la matriz será simétrica.
- La diagonal no puede tener pesos.
- Si no hay aristas conectando ambos nodos, se pone un peso de infinito.



$w(i,j)$ = Peso del arco de V_i a V_j , si existe arco (V_i, V_j)
 ∞ , si NO existe arco (V_i, V_j)
 0, si $V_i = V_j$

M	V1	V2	V3	V4
V1	0	2	3	9
V2	2	0	∞	6
V3	3	∞	0	8
V4	9	6	8	0

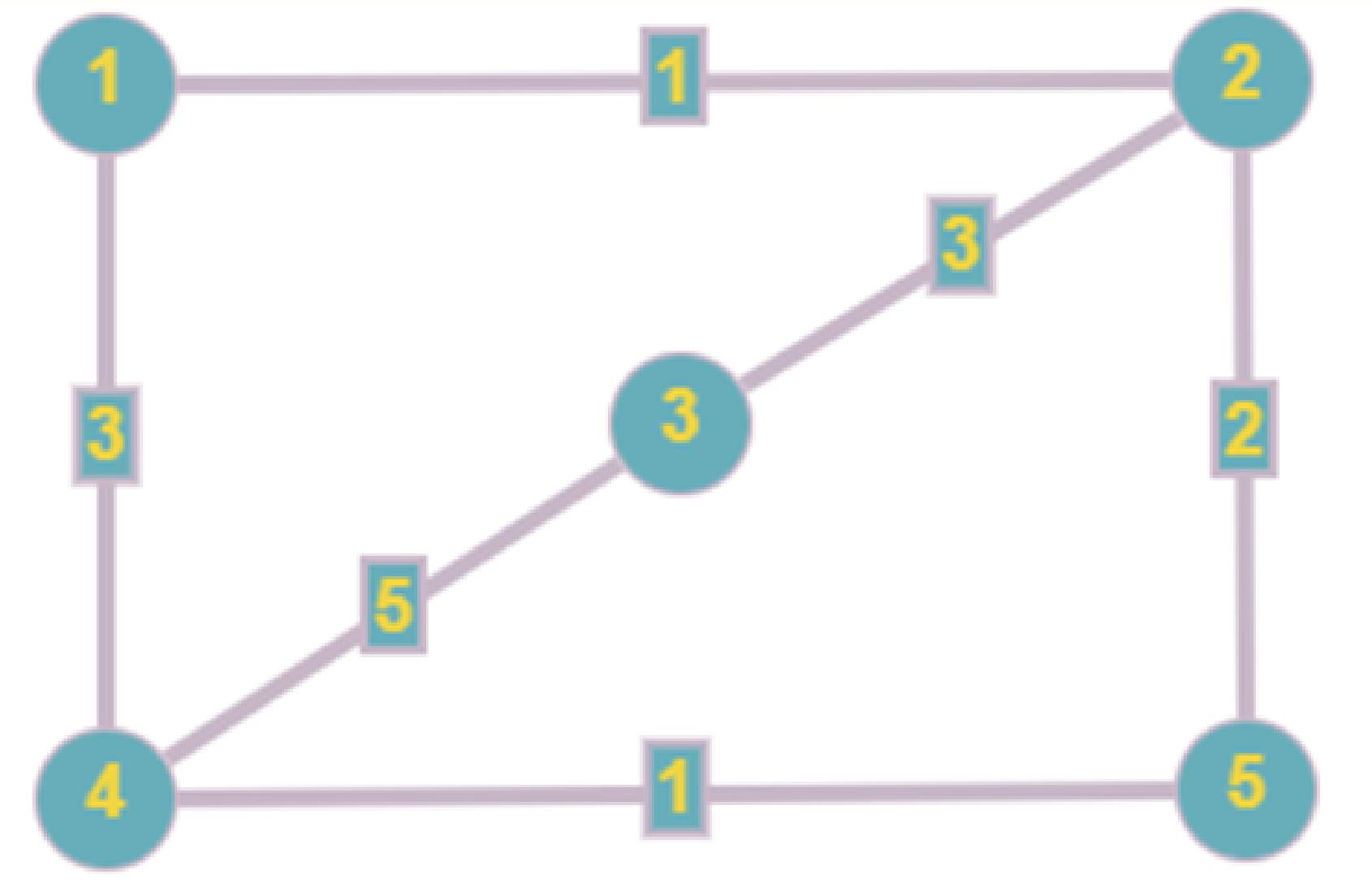
G(V,A)

V:{ V1, V2, V3, V4}

A:{ (V1,V2), (V2,V3), (V3,V4), (V4,V1), (V2,V1), (V3,V2), (V4,V3), (V1,V4), (V1,V3), (V3,V1) }

Ejemplo

Obtener la matriz de adyacencia ponderada del siguiente grafo:



$G(V, A, w)$

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

$A:\{(1,2), (2,3), (3,4), (4,5), (1,4), (2,5), (2,1), (3,2), (4,3), (5,4), (4,1), (5,2)\}$

$w((1,4)) = 3 \quad w((2,5)) = 2 \quad w((2,1)) = 1 \quad w((4,1)) = 3 \quad w((5,2)) = 2 \quad w((1,2)) = 1$

$w((3,4)) = 5 \quad w((3,2)) = 3 \quad w((5,4)) = 1 \quad w((4,3)) = 5 \quad w((2,3)) = 3 \quad w((4,5)) = 1$

Ejemplo

M	1	2	3	4	5
1	0	1	∞	3	∞
2	1	0	3	∞	2
3	∞	3	0	5	∞
4	3	∞	5	0	1
5	∞	2	∞	1	0

$G(V, A, w)$

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

$A:\{(1,2), (2,3), (3,4), (4,5), (1,4), (2,5), (2,1), (3,2), (4,3), (5,4), (4,1), (5,2)\}$

$w((1,4)) = 3 \quad w((2,5)) = 2 \quad w((2,1)) = 1 \quad w((4,1)) = 3 \quad w((5,2)) = 2 \quad w((1,2)) = 1$

$w((3,4)) = 5 \quad w((3,2)) = 3 \quad w((5,4)) = 1 \quad w((4,3)) = 5 \quad w((2,3)) = 3 \quad w((4,5)) = 1$

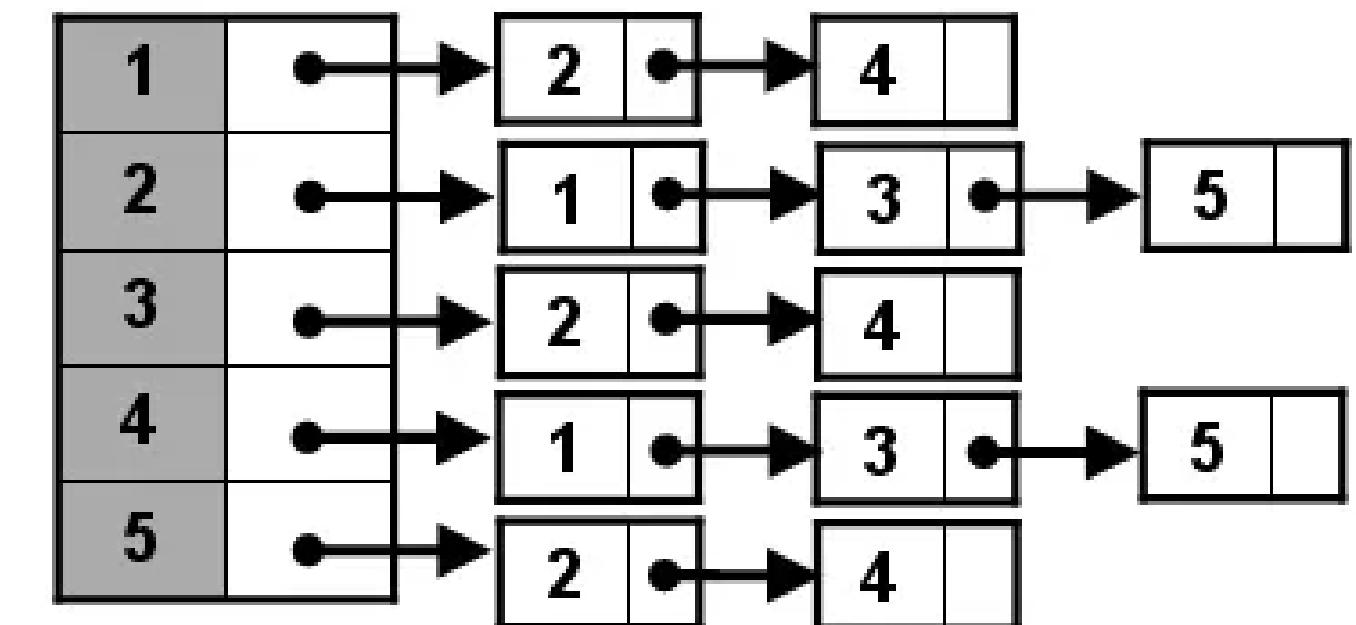
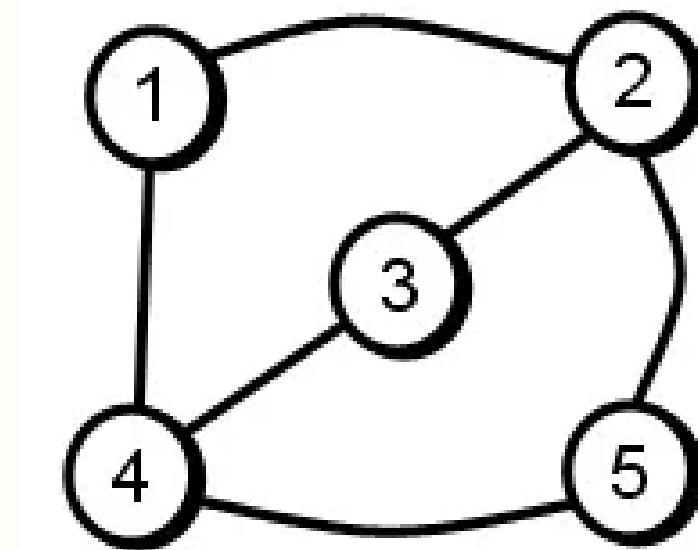
02

Lista de adyacencia:

Cada nodo tiene una lista de nodos a los que es adyacente
Se puede ponderar si además adjuntamos un peso.

Útil para grafos dispersos
(muchos nodos)

Más eficiente en términos de memoria.

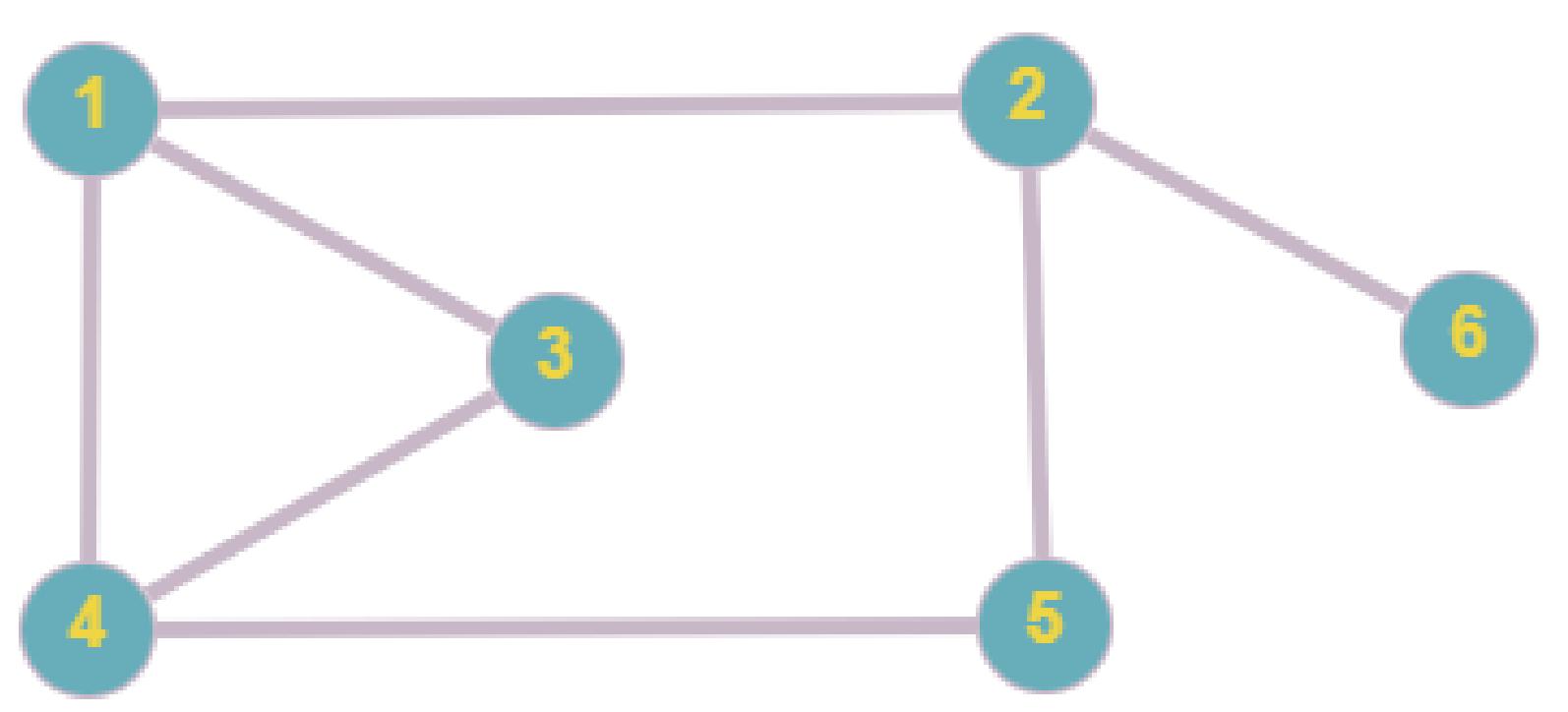


$G(V, A)$

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

$A: \{(1,2), (2,3), (3,4), (4,5), (1,4), (2,5), (2,1), (3,2), (4,3), (5,4), (4,1), (5,2)\}$

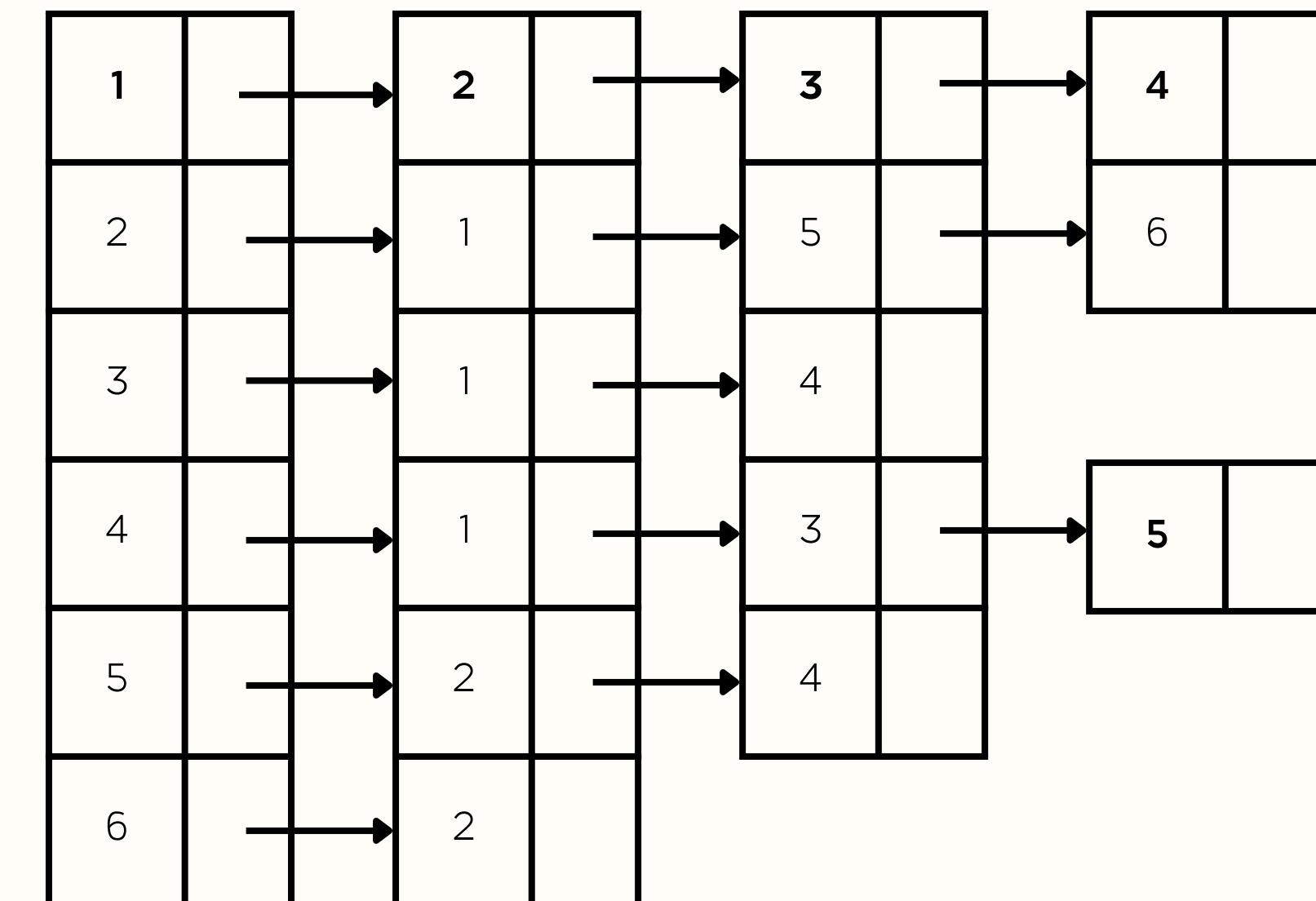
Ejemplo



$G(V, A)$

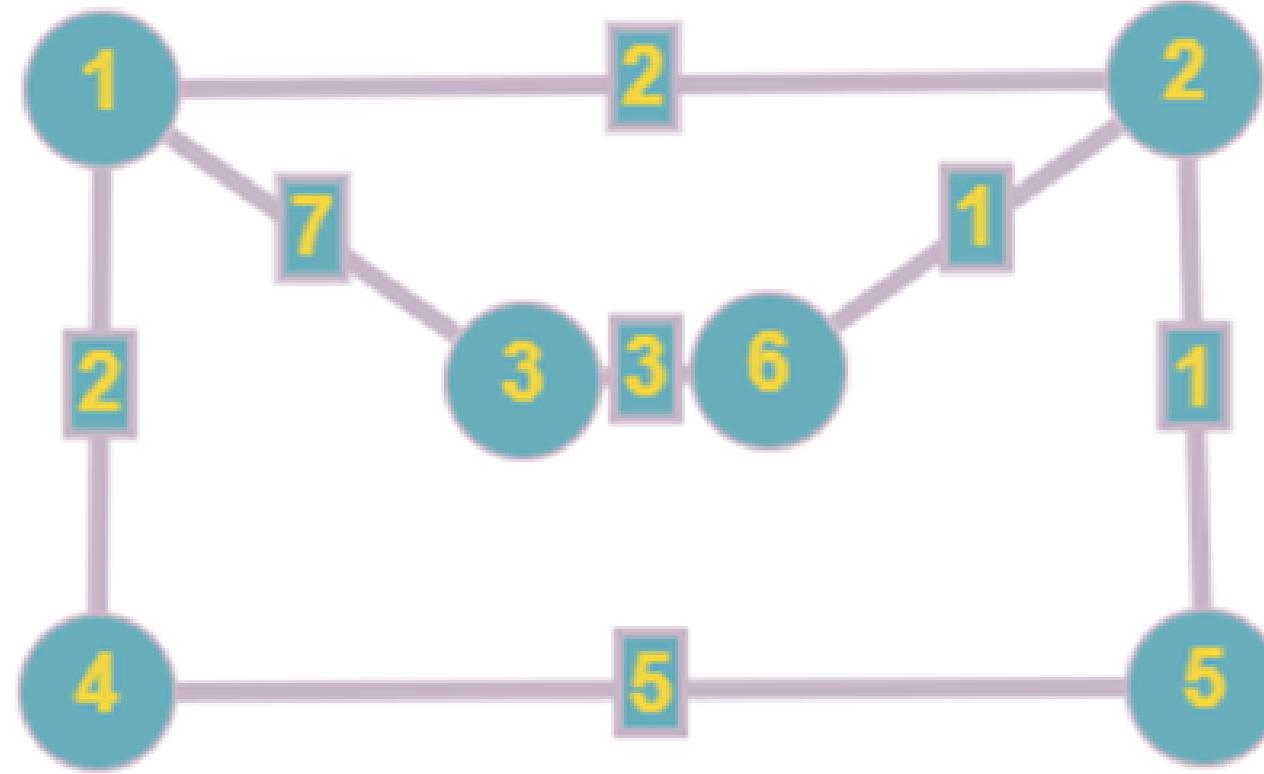
$V: \{1, 2, 3, 4, 5, 6\}$

$A: \{(1,4), (2,5), (2,1), (3,1), (4,3), (5,4), (2,6)\}$



Ejemplo

Lista de adyacencia ponderada



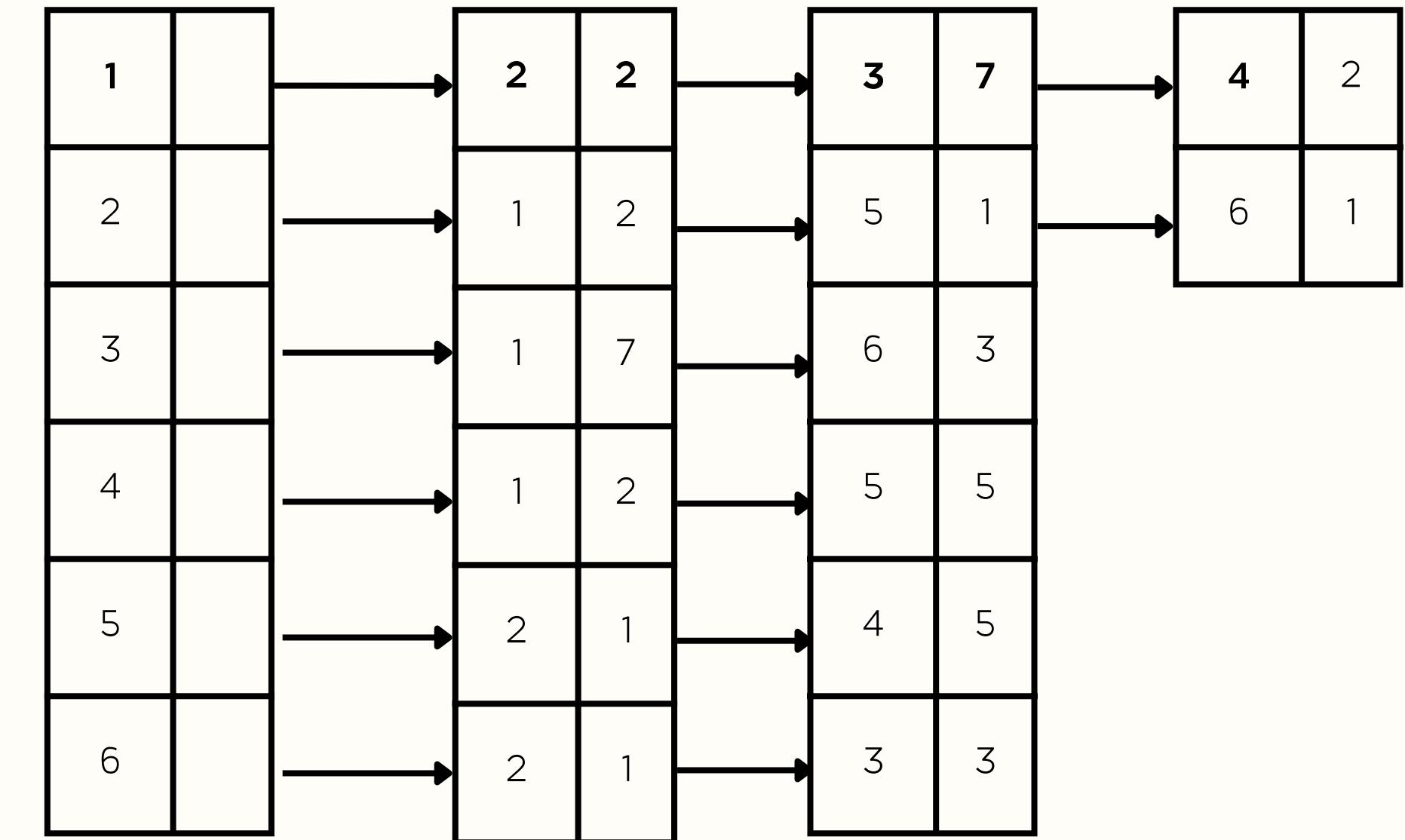
$G(V, A, w)$

$V:\{ 1, 2, 3, 4, 5, 6\}$

$A:\{ (1,4), (2,5), (2,1), (3,1), (3,6), (5,4), (2,6)\}$

$w((1,4)) = 2 \quad w((2,5)) = 1 \quad w((2,1)) = 2 \quad w((2,6)) = 1$

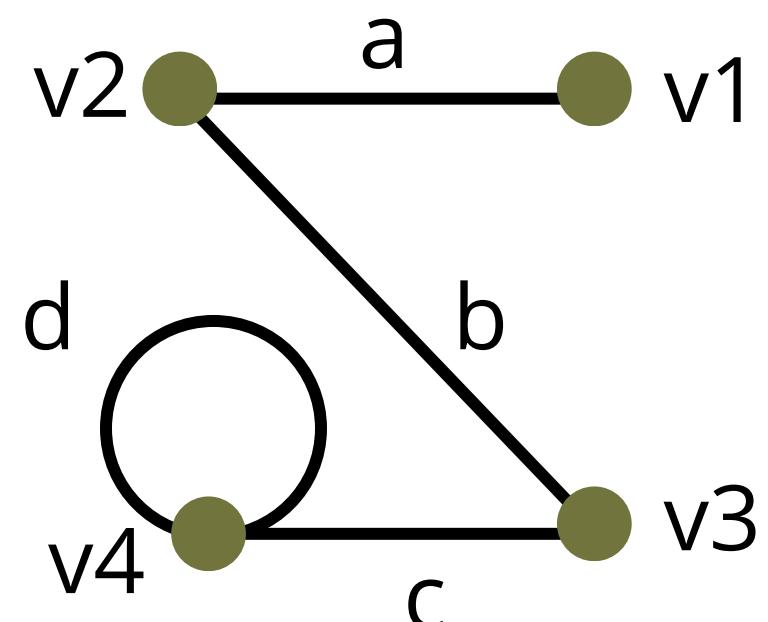
$w((3,1)) = 7 \quad w((3,6)) = 3 \quad w((5,4)) = 5$



03 Matriz de incidencia

Representa la relación entre los vértices y las aristas, muestra como una arista lleva de un nodo a otro.

- Notas:
- Las columnas suman siempre 2.
 - Solo representa grafos no dirigidos en este formato



$G(V, A)$
 $V:\{ V_1, V_2, V_3, V_4\}$
 $A:\{ (a, b, c, d)\}$

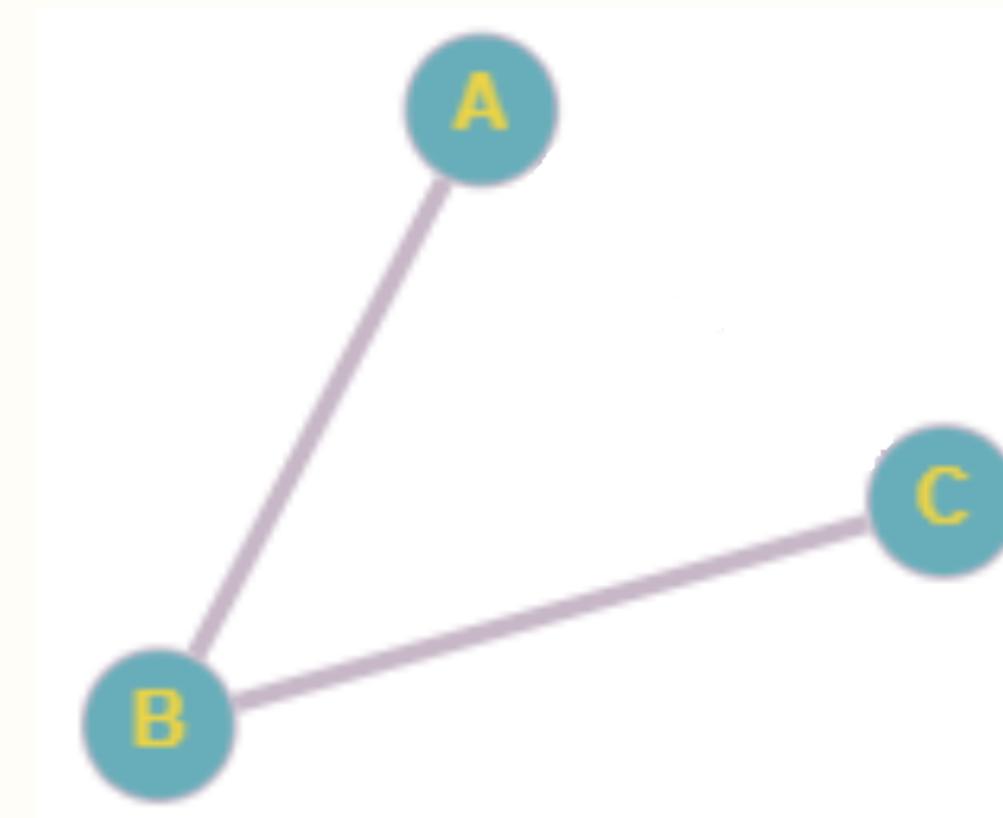
$I(v_i, a_j) = 1$, si el vértice v_i es incidente al arco simple a_j .
2, si el vértice v_i es incidente al lazo a_j .
0, si el vértice v_i no es incidente al arco a_j .

M	a	b	c	d
v1	1	0	0	0
v2	1	1	0	0
v3	0	1	1	0
v4	0	0	1	2

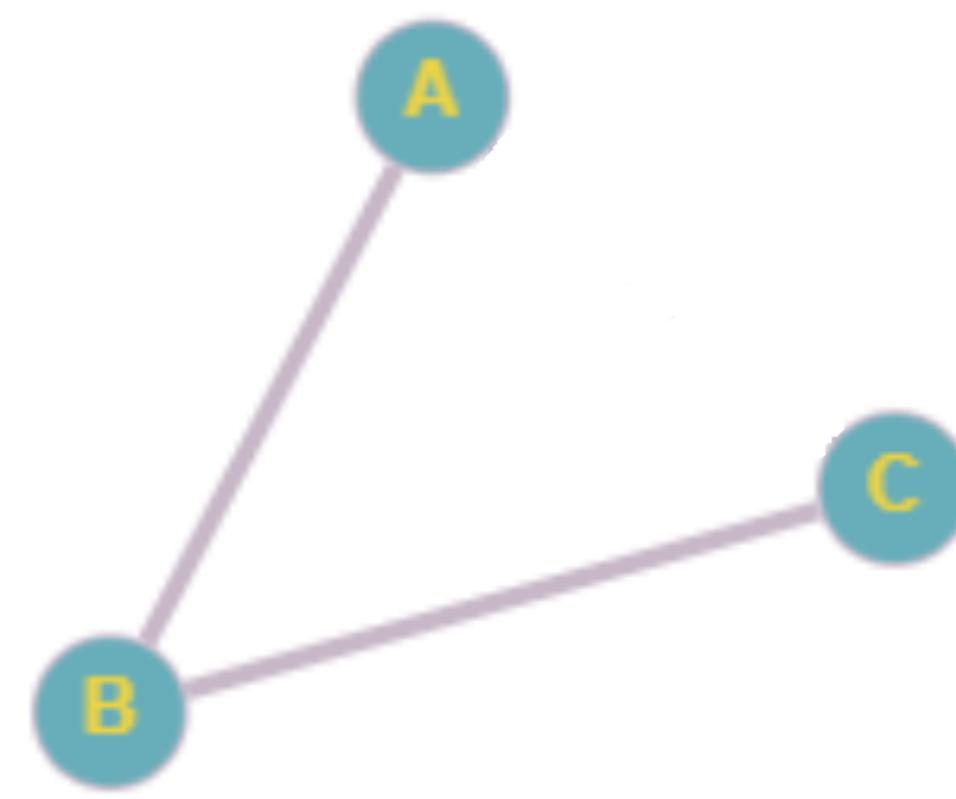
EJERCICIOS

Ejercicio

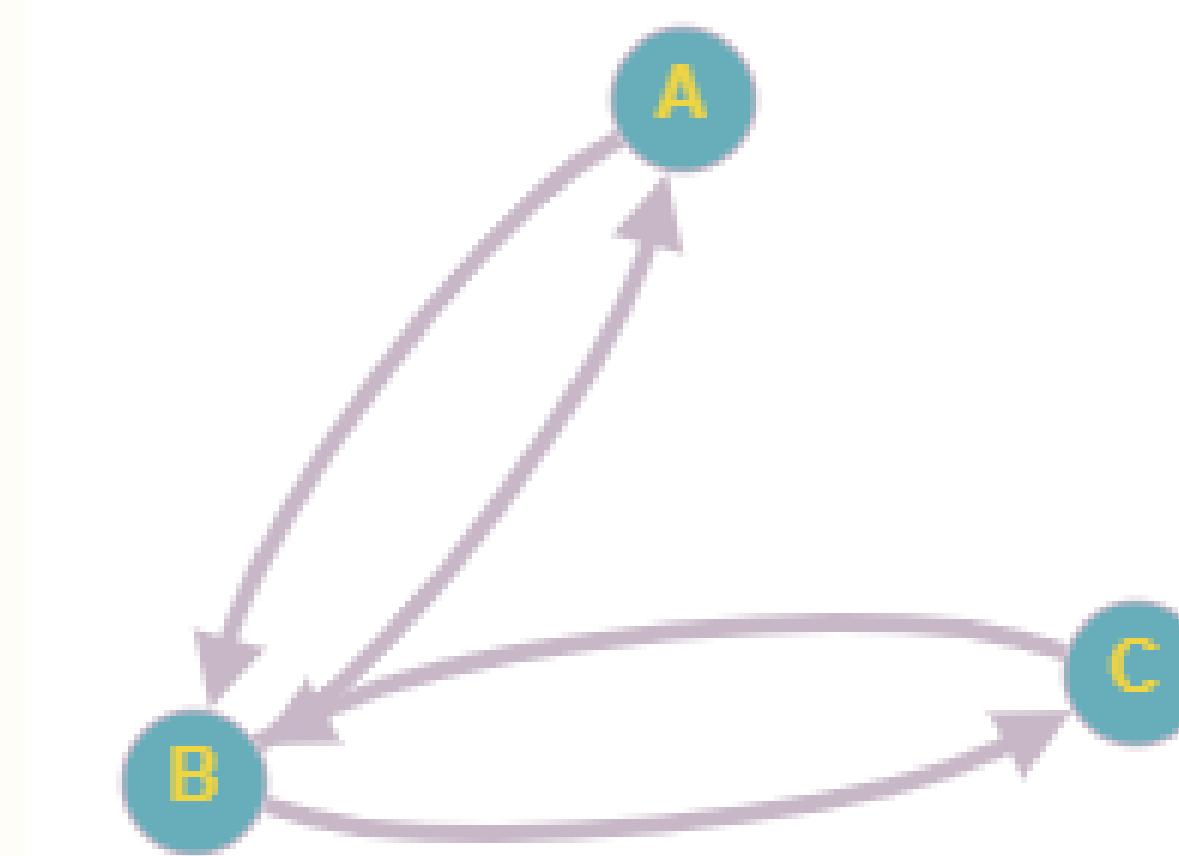
Transformar el grafo no dirigido en grafo dirigido y obtener la representación matemática de ambos



Solución



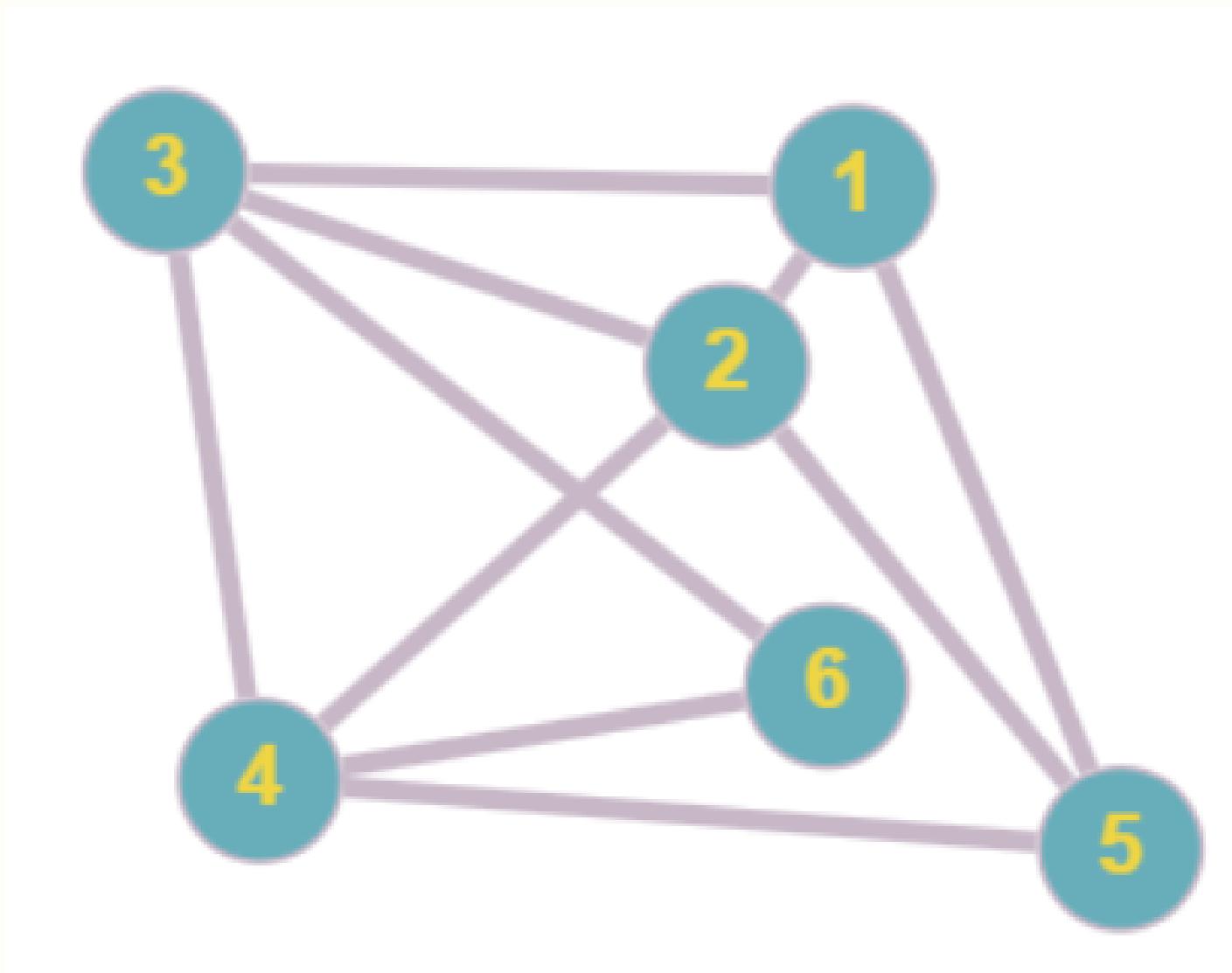
$G(V,A)$
 $V:\{A, B, C\}$
 $A:\{(A,B), (B,C)\}$



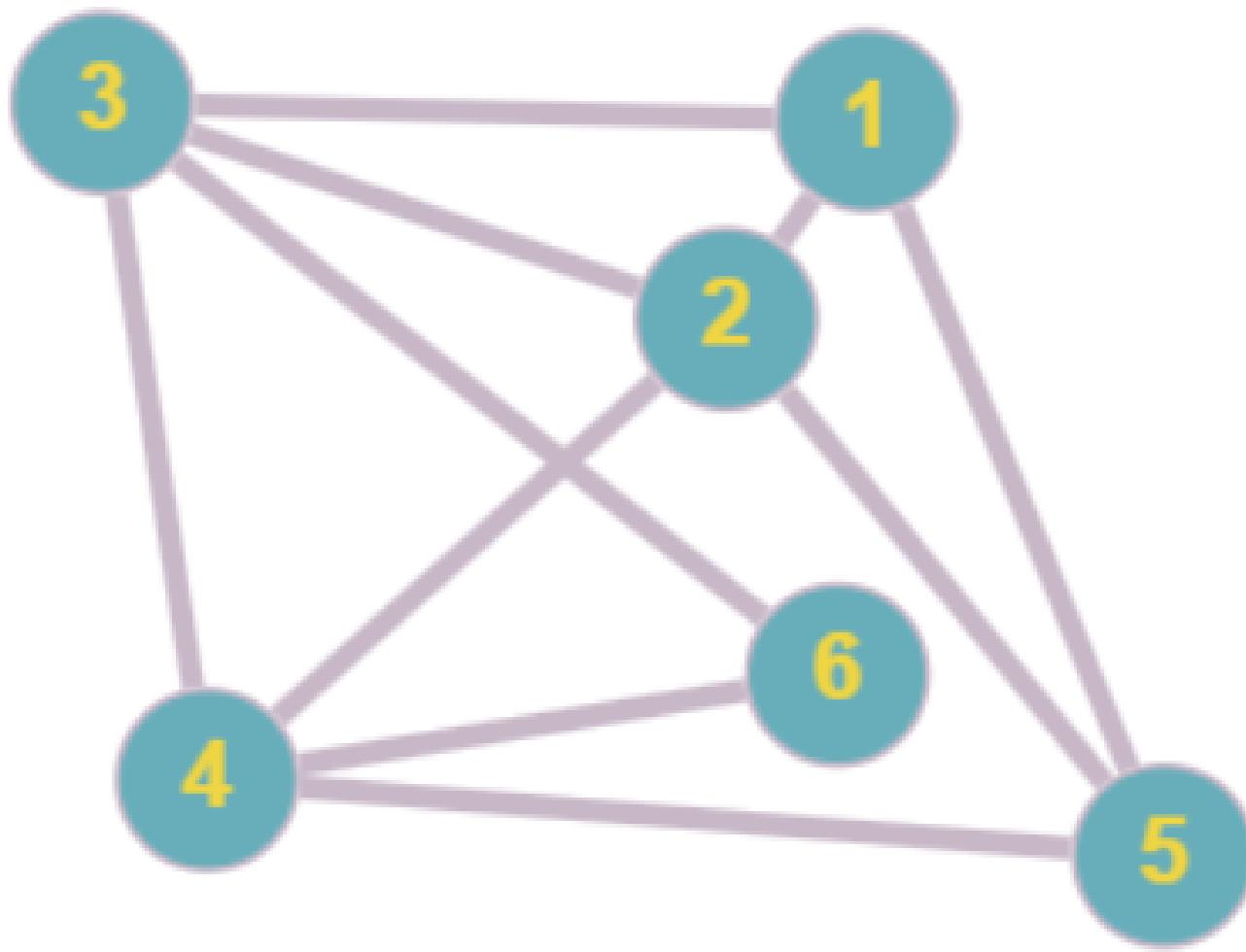
$G(V,A)$
 $V:\{A, B, C\}$
 $A:\{(A,B), (B,A), (B,C), (C,B)\}$

Ejercicio

Obtén la matriz de adyacencia del siguiente grafo no dirigido



Solución



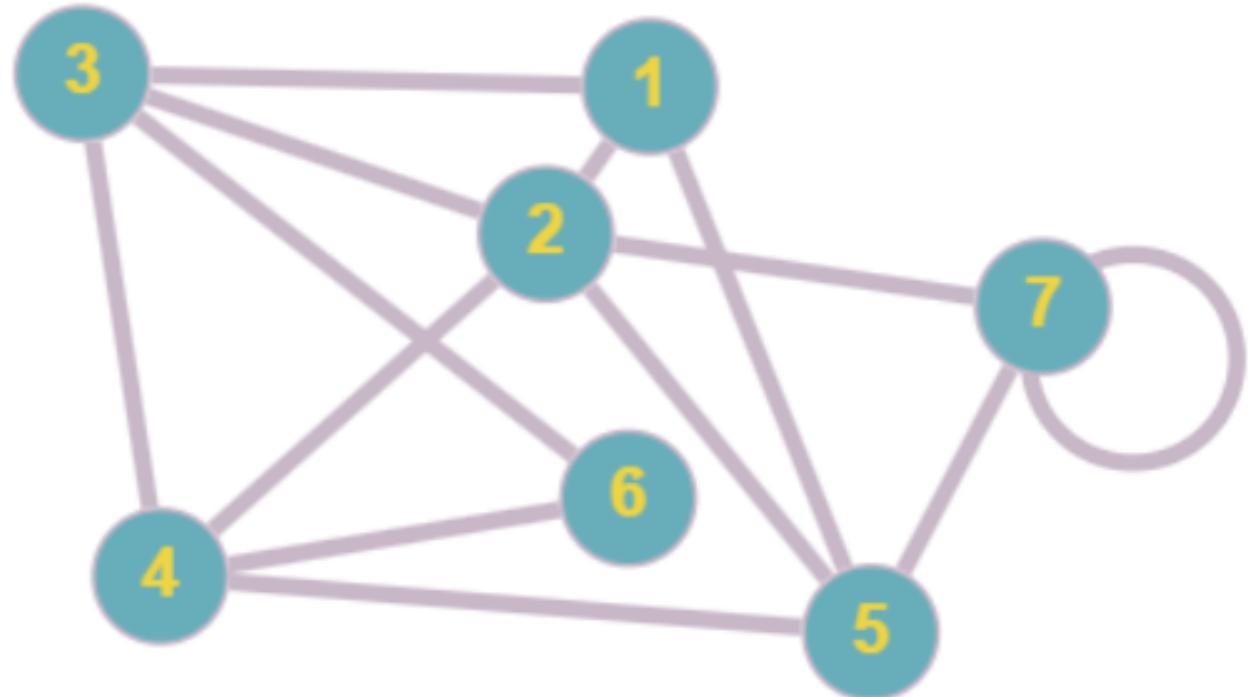
M	V1	V2	V3	V4	V5	V6
V1	0	1	1	0	1	0
V2	1	0	1	1	1	0
V3	1	1	0	1	0	1
V4	0	1	1	0	1	1
V5	1	1	0	1	0	0
V6	0	0	1	1	0	0

Ejercicio

Agregar un vértice v_7 que
contacte con v_2 , v_5 y v_7

M	V1	V2	V3	V4	V5	V6
V1	0	1	1	0	1	0
V2	1	0	1	1	1	0
V3	1	1	0	1	0	1
V4	0	1	1	0	1	1
V5	1	1	0	1	0	0
V6	0	0	1	1	0	0

Solución



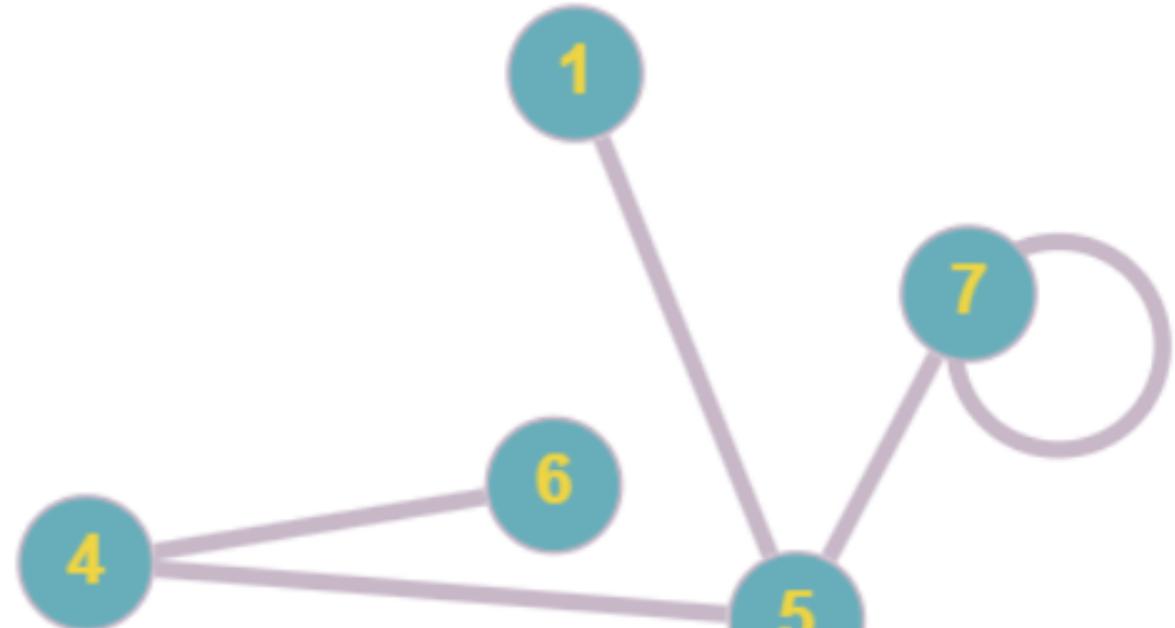
M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	1	0	1	0	0
v2	1	0	1	1	1	0	1
v3	1	1	0	1	0	1	0
v4	0	1	1	0	1	1	0
v5	1	1	0	1	0	0	1
v6	0	0	1	1	0	0	0
v7	0	1	0	0	1	0	2

Ejercicio

**Eliminar el vértice v2
y v3 de la matriz de
adyacencia**

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	1	0	1	0	0
v2	1	0	1	0	1	0	1
v3	1	1	0	1	0	1	0
v4	0	0	1	0	1	1	0
v5	1	1	0	1	0	0	1
v6	0	0	1	1	0	0	0
v7	0	1	0	0	1	0	2

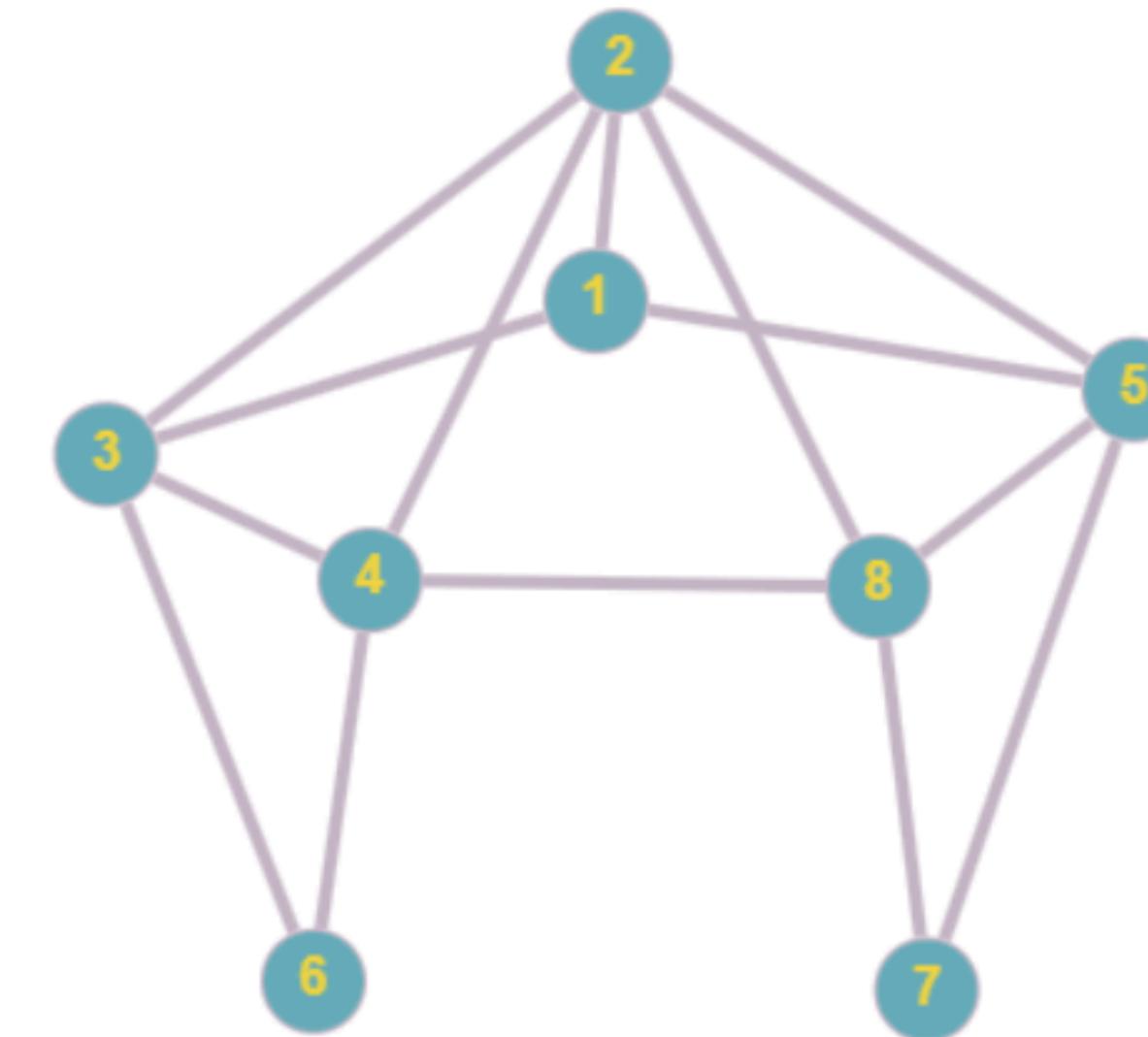
Solución



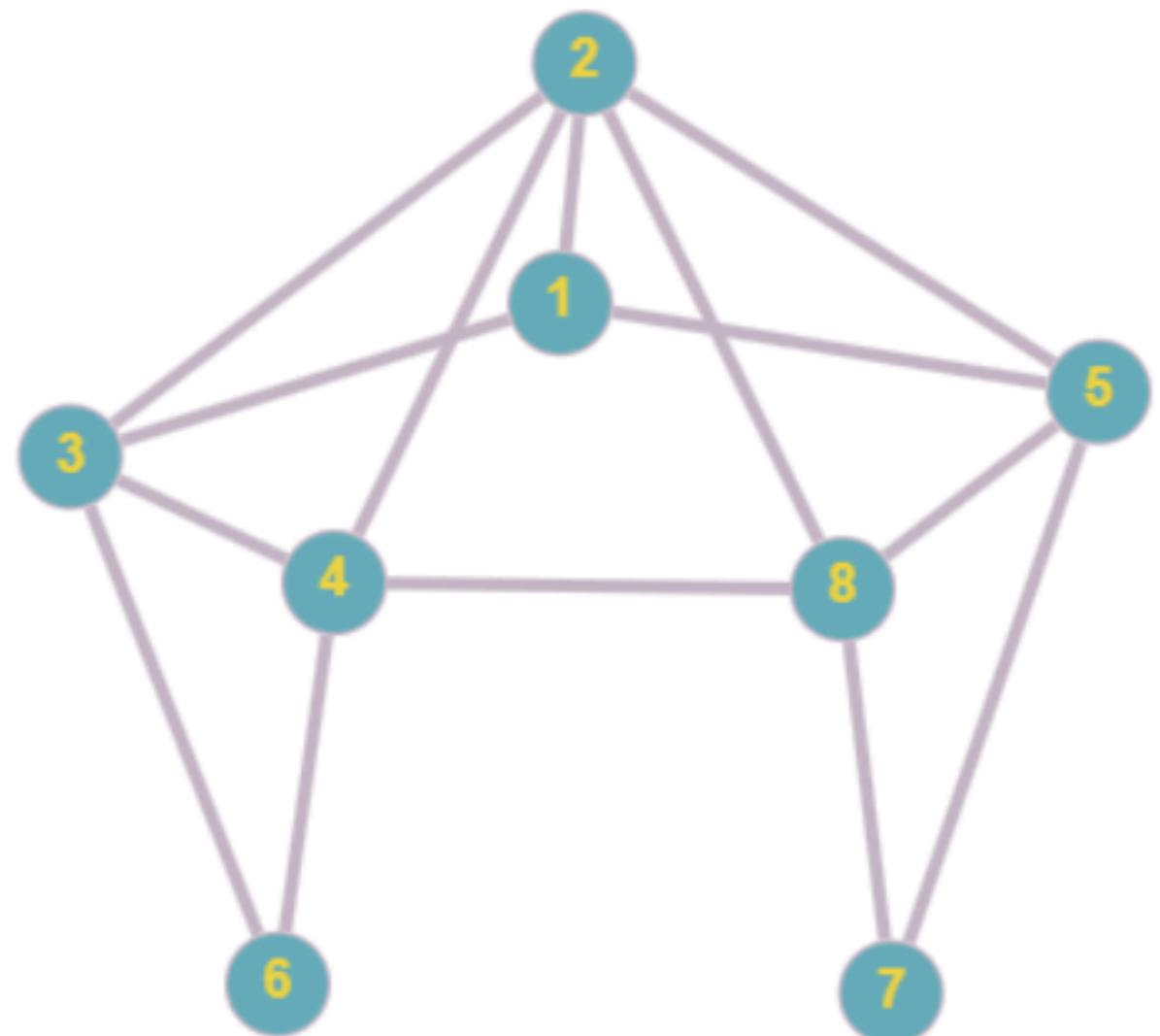
M	v1	v4	v5	v6	v7
v1	0	0	1	0	0
v4	0	0	1	1	0
v5	1	1	0	0	1
v6	0	1	0	0	0
v7	0	0	1	0	2

Ejercicio

Obtén la lista de adyacencia del siguiente grafo no dirigido



Solución



Lista de adyacencia

$v_1 \rightarrow (v_2) \rightarrow (v_3) \rightarrow (v_5)$

$v_2 \rightarrow (v_1) \rightarrow (v_3) \rightarrow (v_4) \rightarrow (v_5) \rightarrow (v_8)$

$v_3 \rightarrow (v_1) \rightarrow (v_2) \rightarrow (v_4) \rightarrow (v_6)$

$v_4 \rightarrow (v_2) \rightarrow (v_3) \rightarrow (v_6) \rightarrow (v_8)$

$v_5 \rightarrow (v_1) \rightarrow (v_2) \rightarrow (v_7) \rightarrow (v_8)$

$v_6 \rightarrow (v_3) \rightarrow (v_4)$

$v_7 \rightarrow (v_5) \rightarrow (v_8)$

$v_8 \rightarrow (v_2) \rightarrow (v_4) \rightarrow (v_5) \rightarrow (v_7)$

Ejercicio

Agrega el vértice v9 que conecte con v7, v8, v3 y v1

Lista de adyacencia

v1 → (v2) → (v3) → (v5)

v2 → (v1) → (v3) → (v4) → (v5) → (v8)

v3 → (v1) → (v2) → (v4) → (v6)

v4 → (v2) → (v3) → (v6) → (v8)

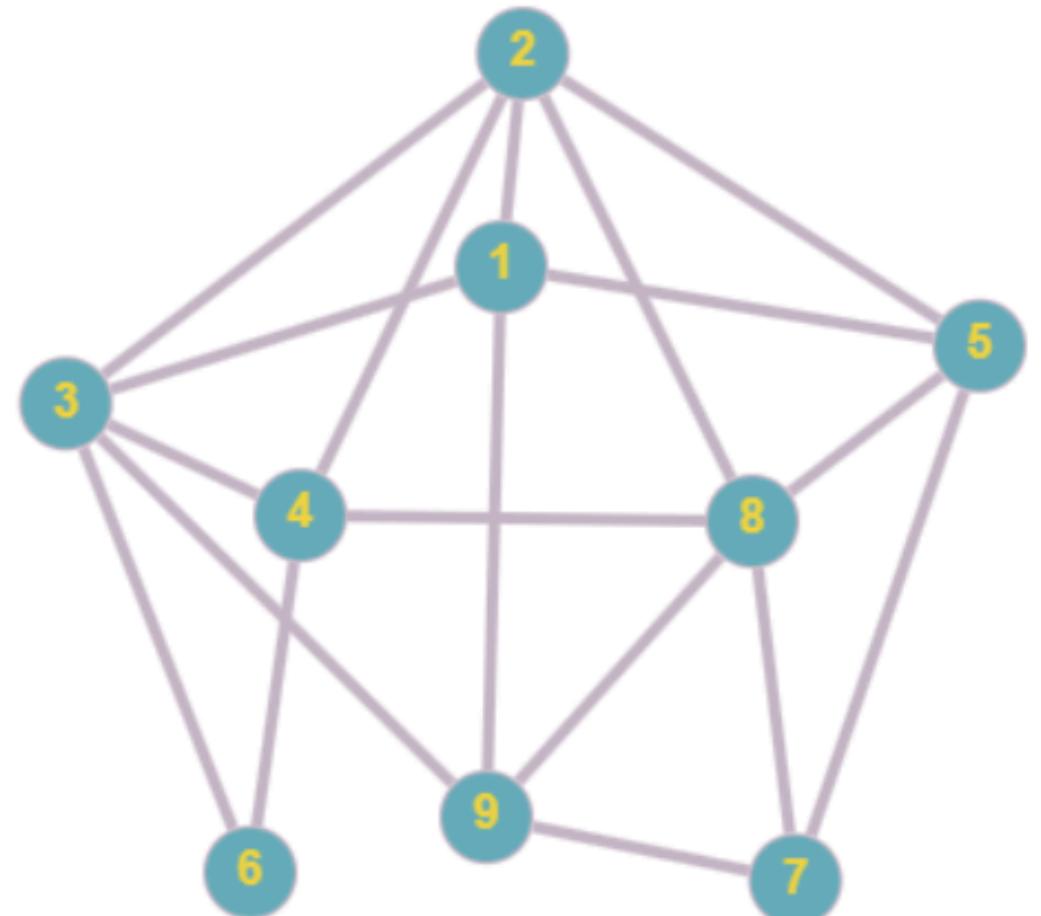
v5 → (v1) → (v2) → (v7) → (v8)

v6 → (v3) → (v4)

v7 → (v5) → (v8)

v8 → (v2) → (v4) → (v5) → (v7)

Solución



Lista de adyacencia

$v1 \rightarrow (v2) \rightarrow (v3) \rightarrow (v5) \rightarrow (v9)$
 $v2 \rightarrow (v1) \rightarrow (v3) \rightarrow (v4) \rightarrow (v5) \rightarrow (v8)$
 $v3 \rightarrow (v1) \rightarrow (v2) \rightarrow (v4) \rightarrow (v6) \rightarrow (v9)$
 $v4 \rightarrow (v2) \rightarrow (v3) \rightarrow (v6) \rightarrow (v8)$
 $v5 \rightarrow (v1) \rightarrow (v2) \rightarrow (v7) \rightarrow (v8)$
 $v6 \rightarrow (v3) \rightarrow (v4)$
 $v7 \rightarrow (v5) \rightarrow (v8) \rightarrow (v9)$
 $v8 \rightarrow (v2) \rightarrow (v4) \rightarrow (v5) \rightarrow (v7) \rightarrow (v9)$
 $v9 \rightarrow (v1) \rightarrow (v3) \rightarrow (v7) \rightarrow (v8)$

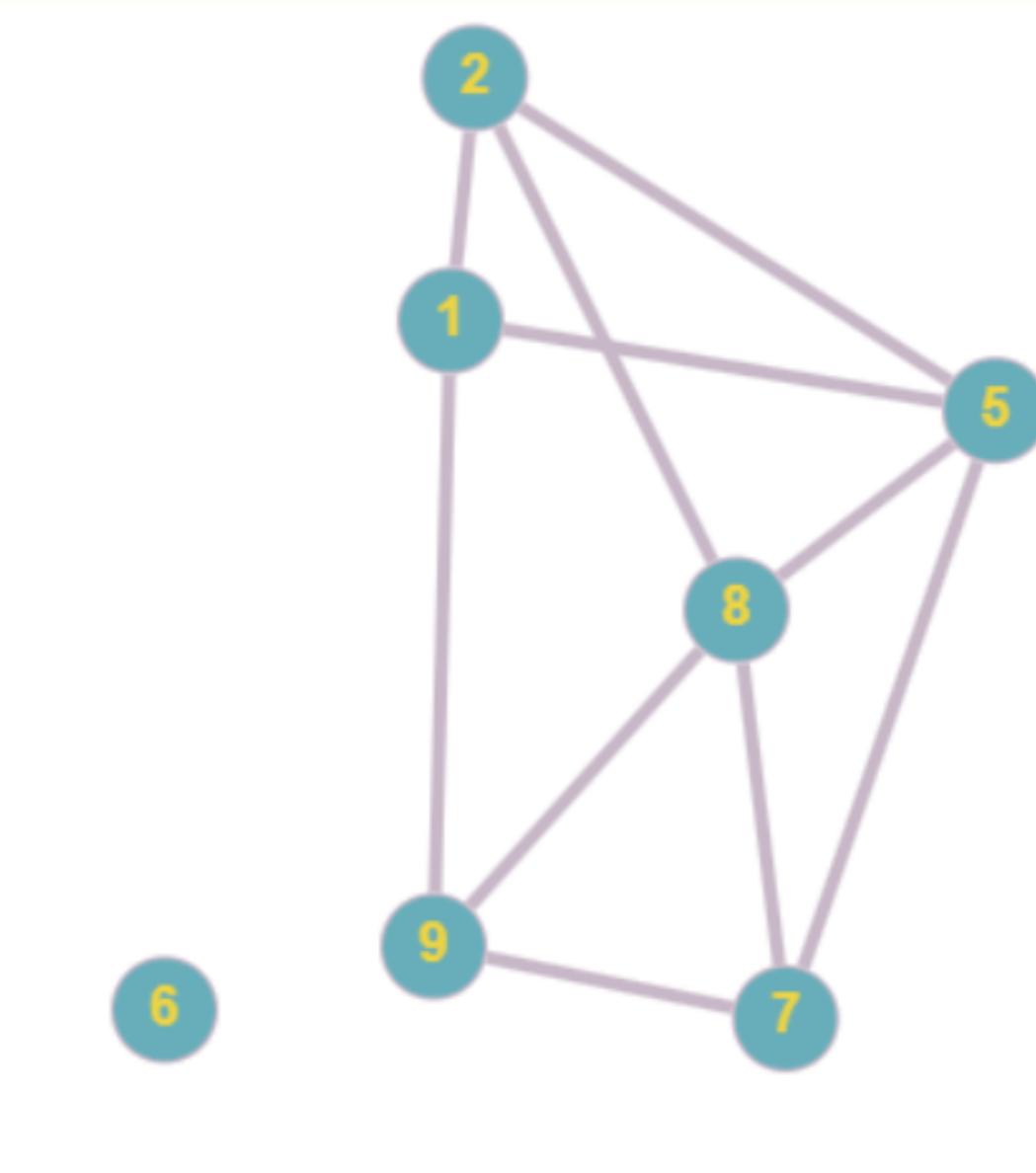
Ejercicio

**Eliminar el vértice v3 y v4
de la lista de adyacencia**

Lista de adyacencia

v1 → (v2) → (v3) → (v5) → (v9)
v2 → (v1) → (v3) → (v4) → (v5) → (v8)
v3 → (v1) → (v2) → (v4) → (v6) → (v9)
v4 → (v2) → (v3) → (v6) → (v8)
v5 → (v1) → (v2) → (v7) → (v8)
v6 → (v3) → (v4)
v7 → (v5) → (v8) → (v9)
v8 → (v2) → (v4) → (v5) → (v7) → (v9)
v9 → (v1) → (v3) → (v7) → (v8)

Solución



Lista de adyacencia

$v1 \rightarrow (v2) \rightarrow (v5) \rightarrow (v9)$

$v2 \rightarrow (v1) \rightarrow (v5) \rightarrow (v8)$

$v5 \rightarrow (v1) \rightarrow (v2) \rightarrow (v7) \rightarrow (v8)$

$v6$

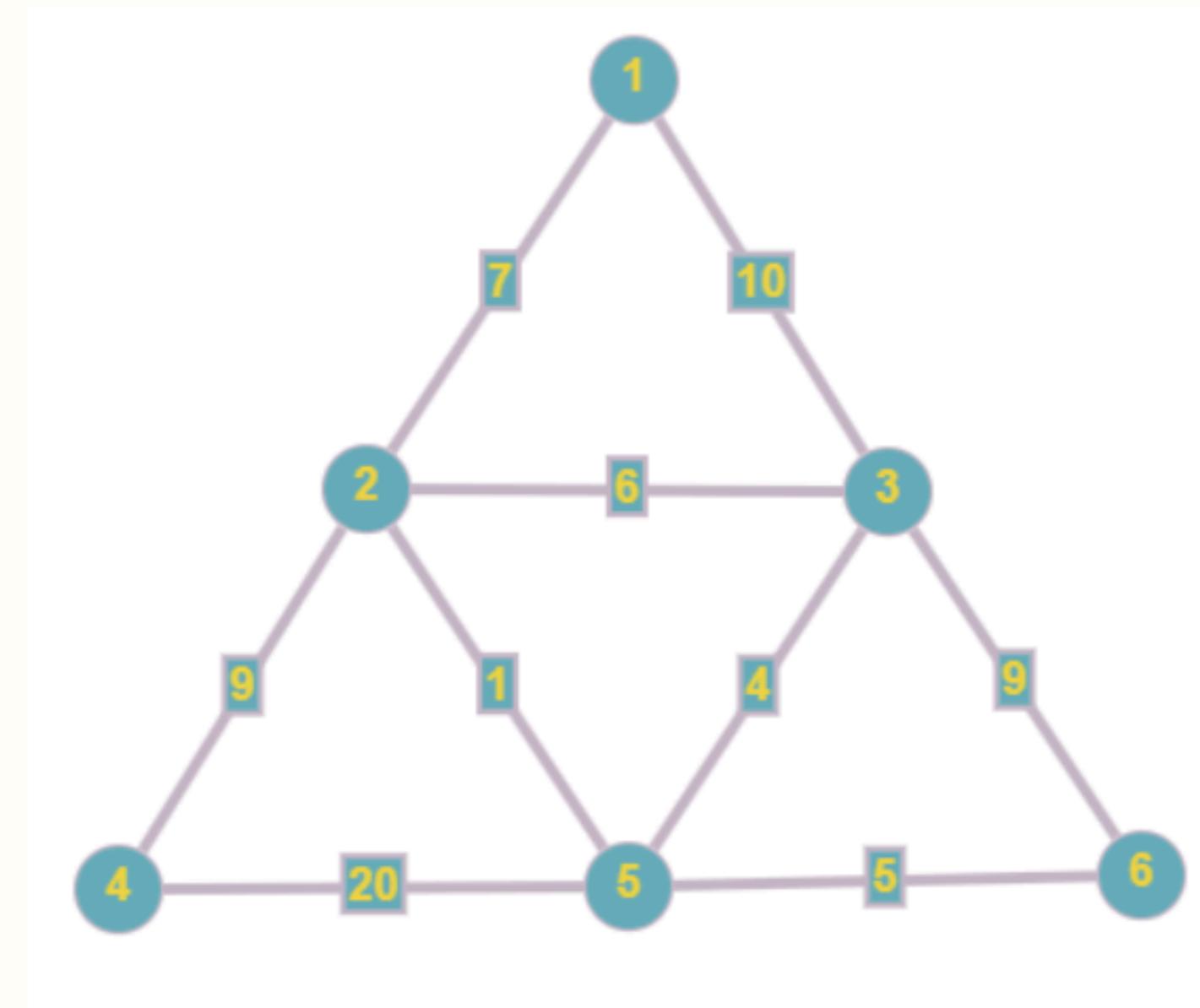
$v7 \rightarrow (v5) \rightarrow (v8) \rightarrow (v9)$

$v8 \rightarrow (v2) \rightarrow (v5) \rightarrow (v7) \rightarrow (v9)$

$v9 \rightarrow (v1) \rightarrow (v7) \rightarrow (v8)$

Ejercicio

Obtén la lista y matriz de adyacencia ponderada del siguiente grafo



Solución

Obtén la lista y matriz de adyacencia ponderada del siguiente grafo

M	v1	v2	v3	v4	v5	v6
v1	0	7	10	∞	∞	∞
v2	7	0	6	9	1	∞
v3	10	6	0	∞	4	9
v4	∞	9	∞	0	20	∞
v5	∞	1	4	20	0	5
v6	∞	∞	9	∞	5	0

Lista de adyacencia:

$v1 \rightarrow (v2, 7) \rightarrow (v3, 10)$

$v2 \rightarrow (v1, 7) \rightarrow (v3, 6) \rightarrow (v4, 9) \rightarrow (v5, 1)$

$v3 \rightarrow (v1, 10) \rightarrow (v2, 6) \rightarrow (v5, 4) \rightarrow (v6, 9)$

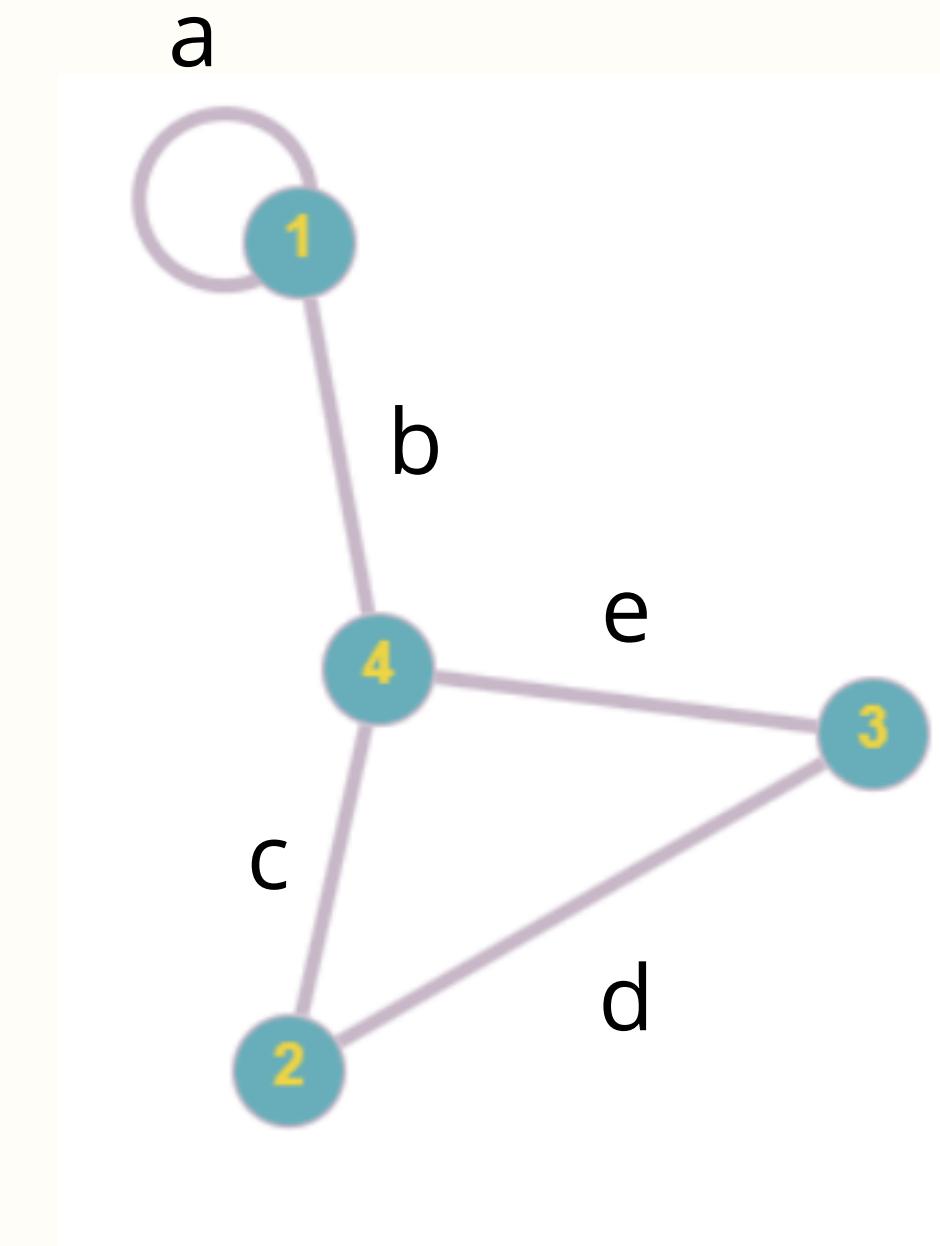
$v4 \rightarrow (v2, 9) \rightarrow (v5, 20)$

$v5 \rightarrow (v2, 1) \rightarrow (v3, 4) \rightarrow (v4, 20) \rightarrow (v6, 5)$

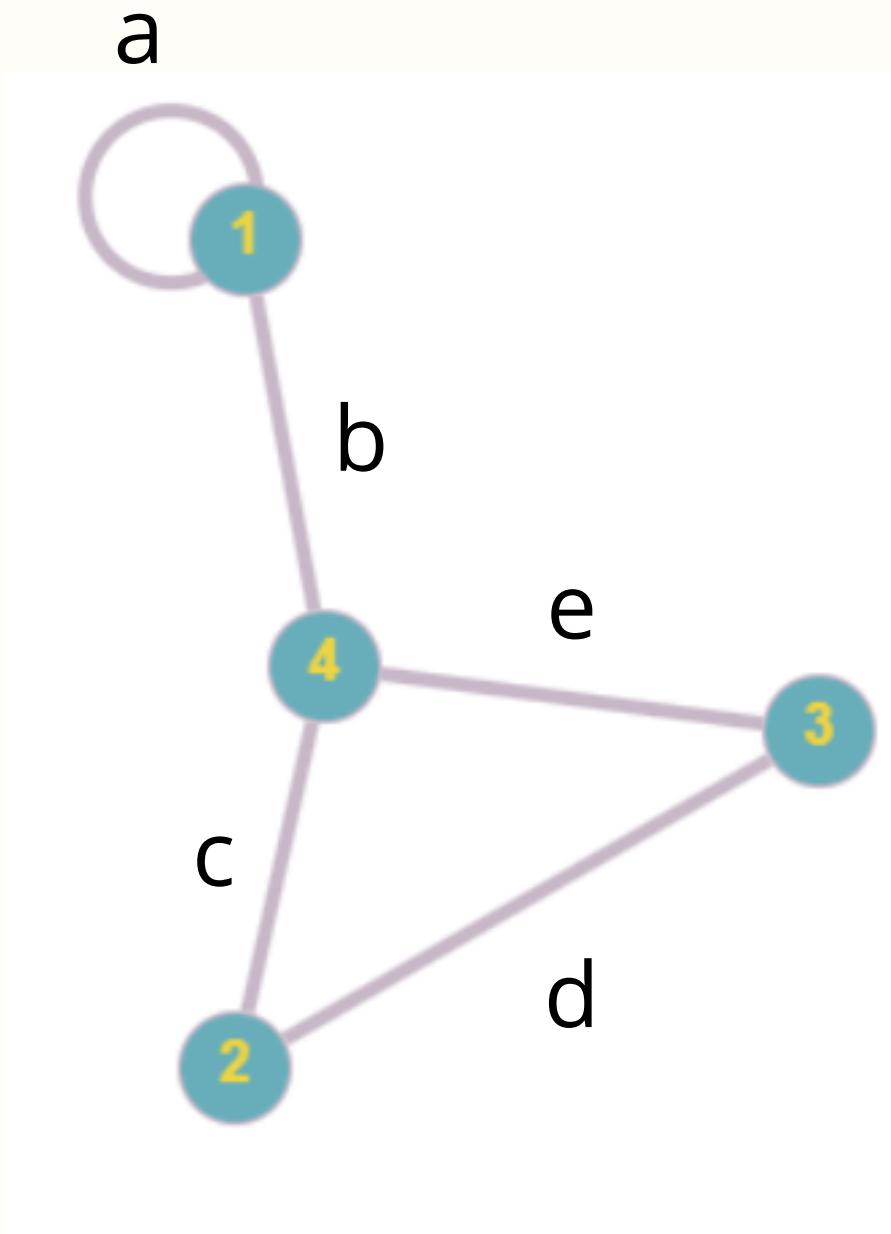
$v6 \rightarrow (v3, 9) \rightarrow (v5, 5)$

Ejercicio

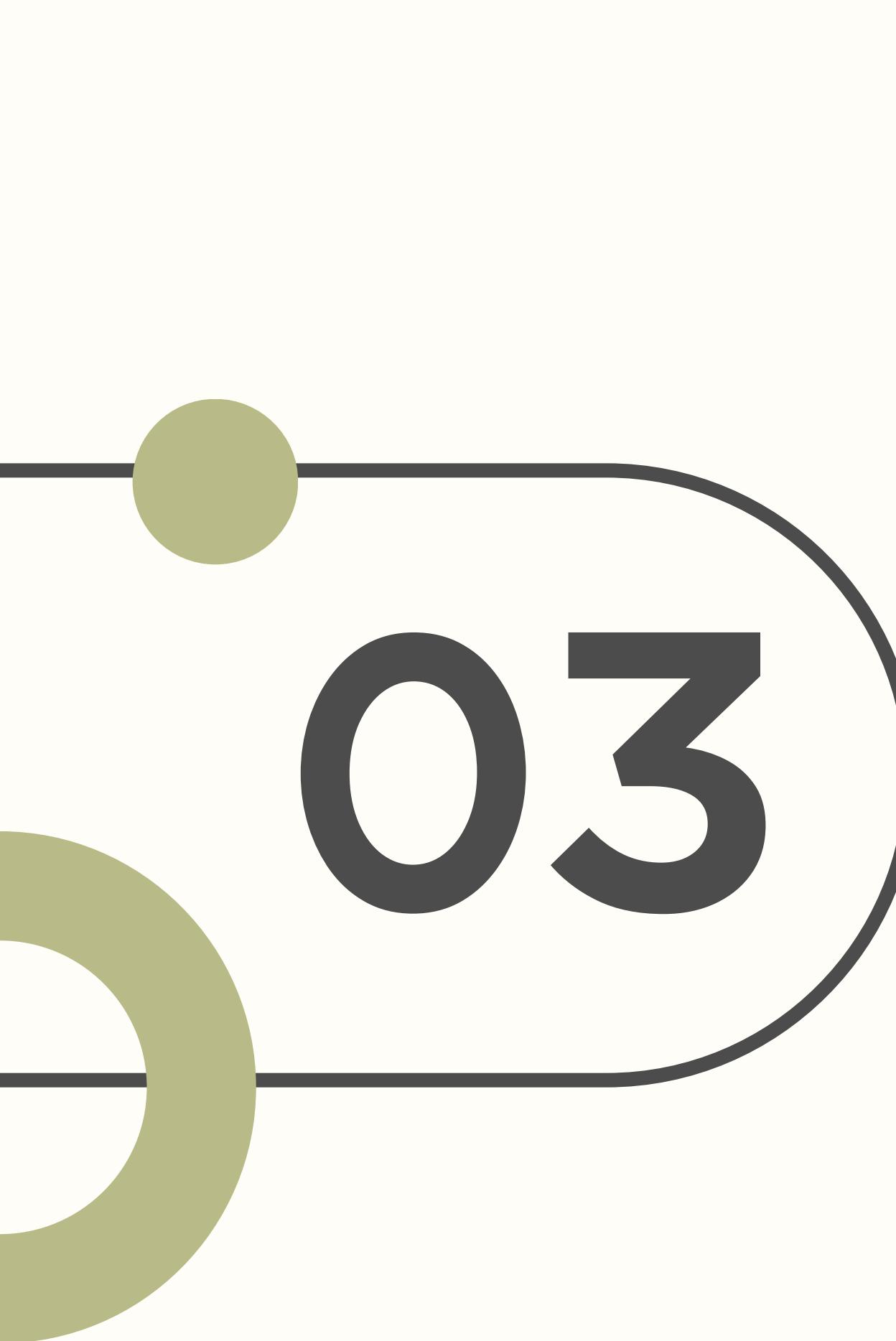
Obten la matriz de incidencia del siguiente grafo no dirigido



Solución



M	a	b	c	d	e
v1	2	1	0	0	0
v2	0	0	1	1	0
v3	0	0	0	1	1
v4	0	1	1	0	1



03

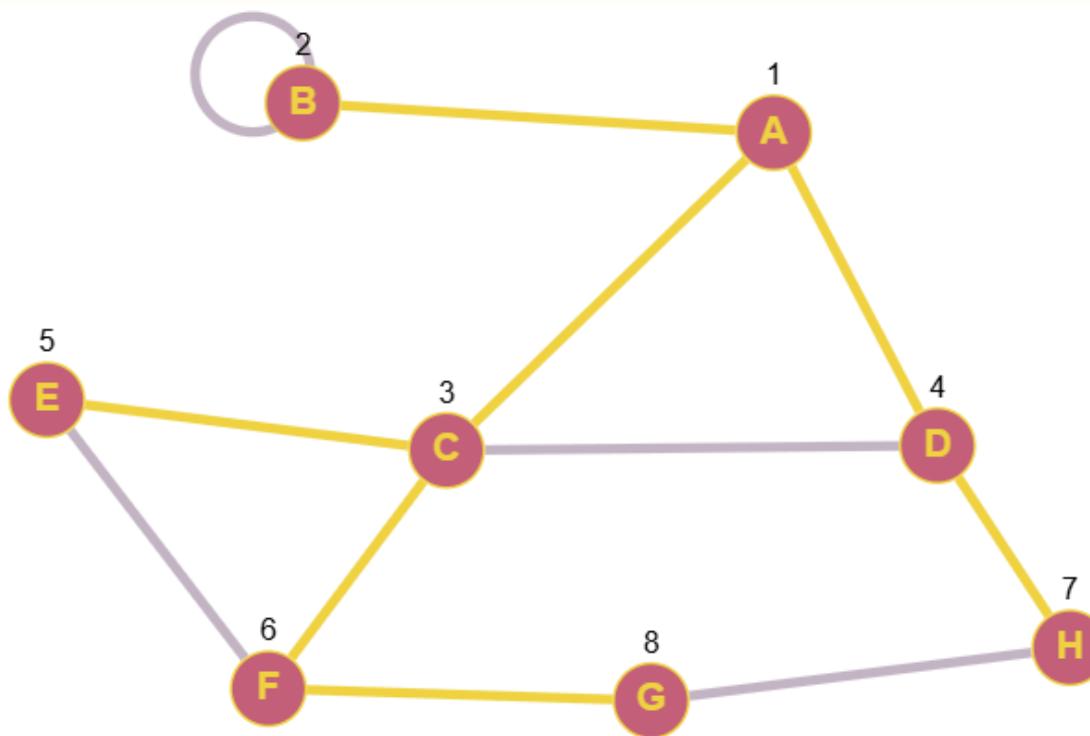
**Recorridos de un
grafo (Anchura y
Profundidad)**

Recorridos de un grafo

Sirven para construir árboles de expansión del grafo con el objetivo de visitar todos los nodos posibles.

Se puede recorrer por anchura y por profundidad.

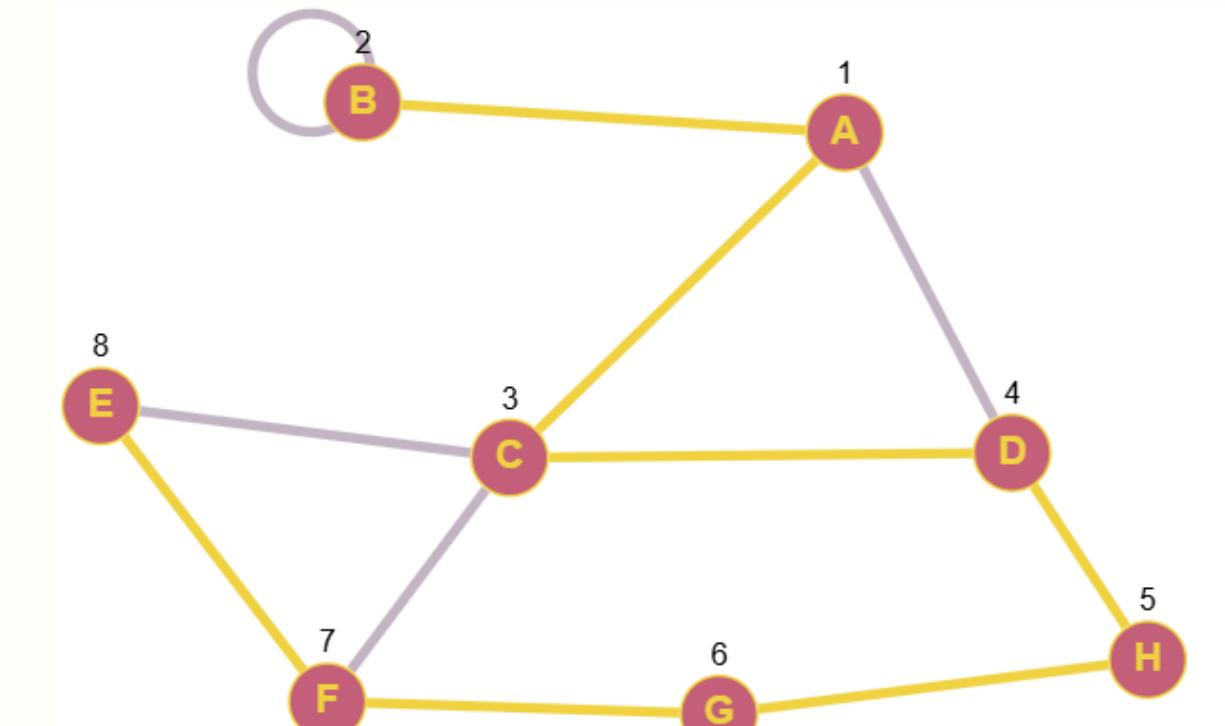
Recorrido en anchura



Notas:

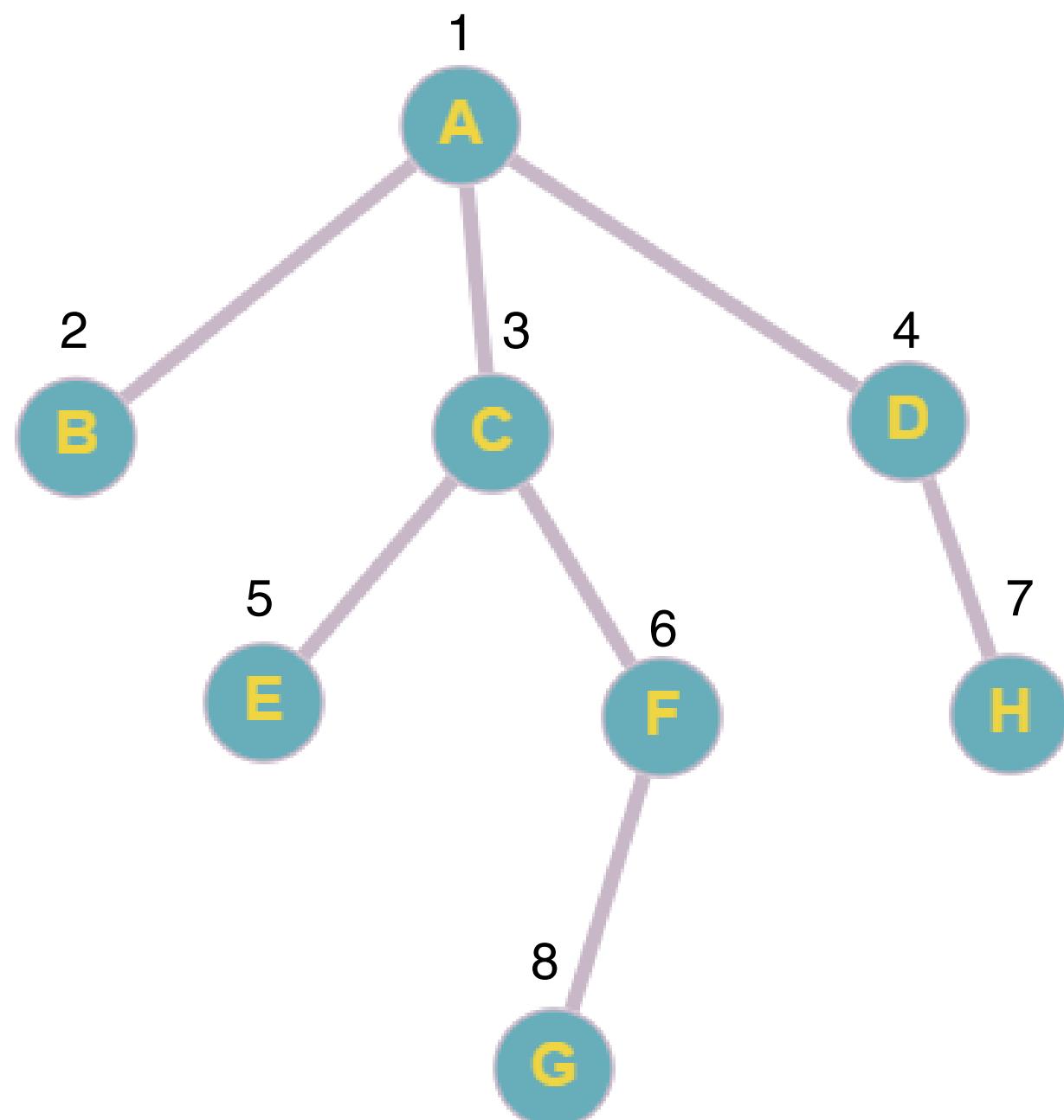
- Un árbol generador es un subgrafo con todos los vértices del grafo original que además es un árbol.

Recorrido en profundidad

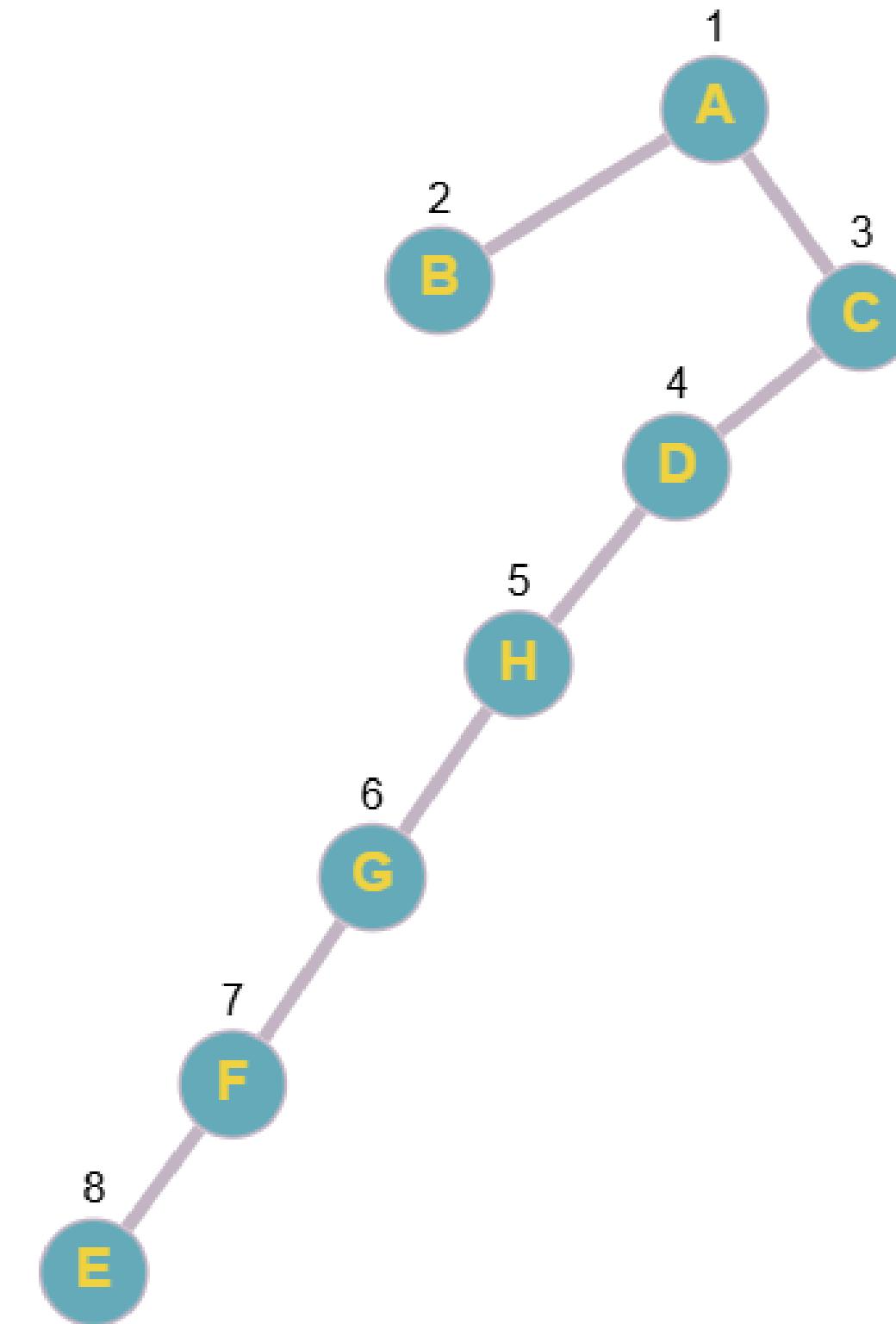


Arboles

Recorrido en anchura



Recorrido en profundidad



→ ANCHURA

Un recorrido por anchura es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo, comenzando en la raíz (eliendo algún nodo como elemento raíz en el caso de un grafo), para luego explorar todos los vecinos de este nodo. A continuación, para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo.

BFS: Breadth-First Search

→ ANCHURA

Recorrido por anchura

- Comienza de un nodo elegido arbitrariamente llamado nodo raíz
- Cada vecino de este nodo se visita en orden; de no estar en la lista se almacena en ella y en la cola.
- Se saca un nodo de la cola, se elige como nuevo nodo raíz y se repite el proceso.
- Termina cuando no hay elementos en la cola al momento de sacar.

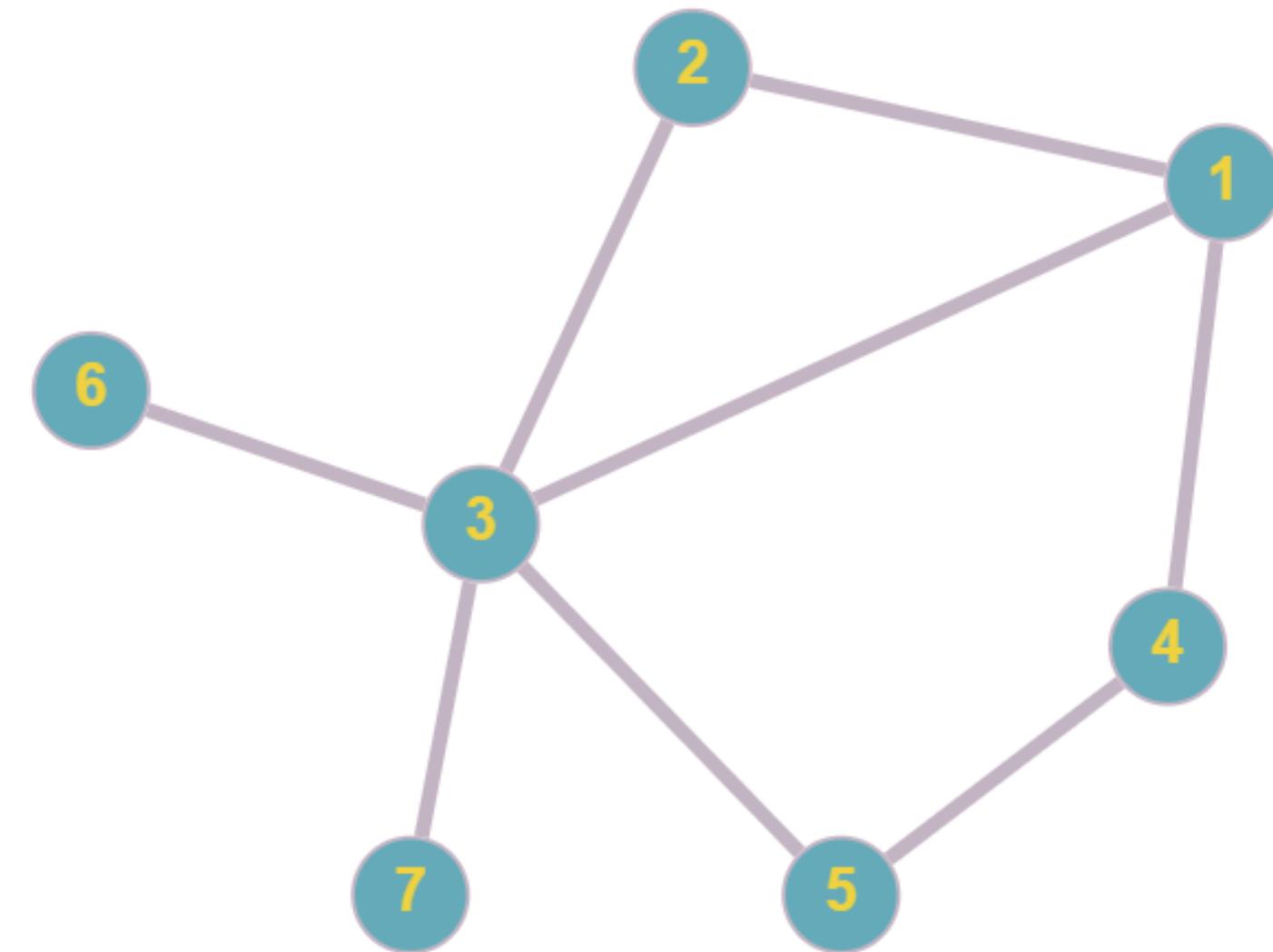
Notas:

- La lista empezará con el nodo raíz en la primera iteración
- La cola empezará vacía
- Si se quiere construir el árbol de expansión, se incluye al primer nodo del cual empezó el algoritmo y cada vez que se guarda en la cola se agrega al árbol de expansión así como la arista de donde vino

Utilizado para calcular la ruta más eficiente a un nodo

Anchura

Se desea realizar un recorrido por anchura comenzado desde el nodo 1 del siguiente grafo:

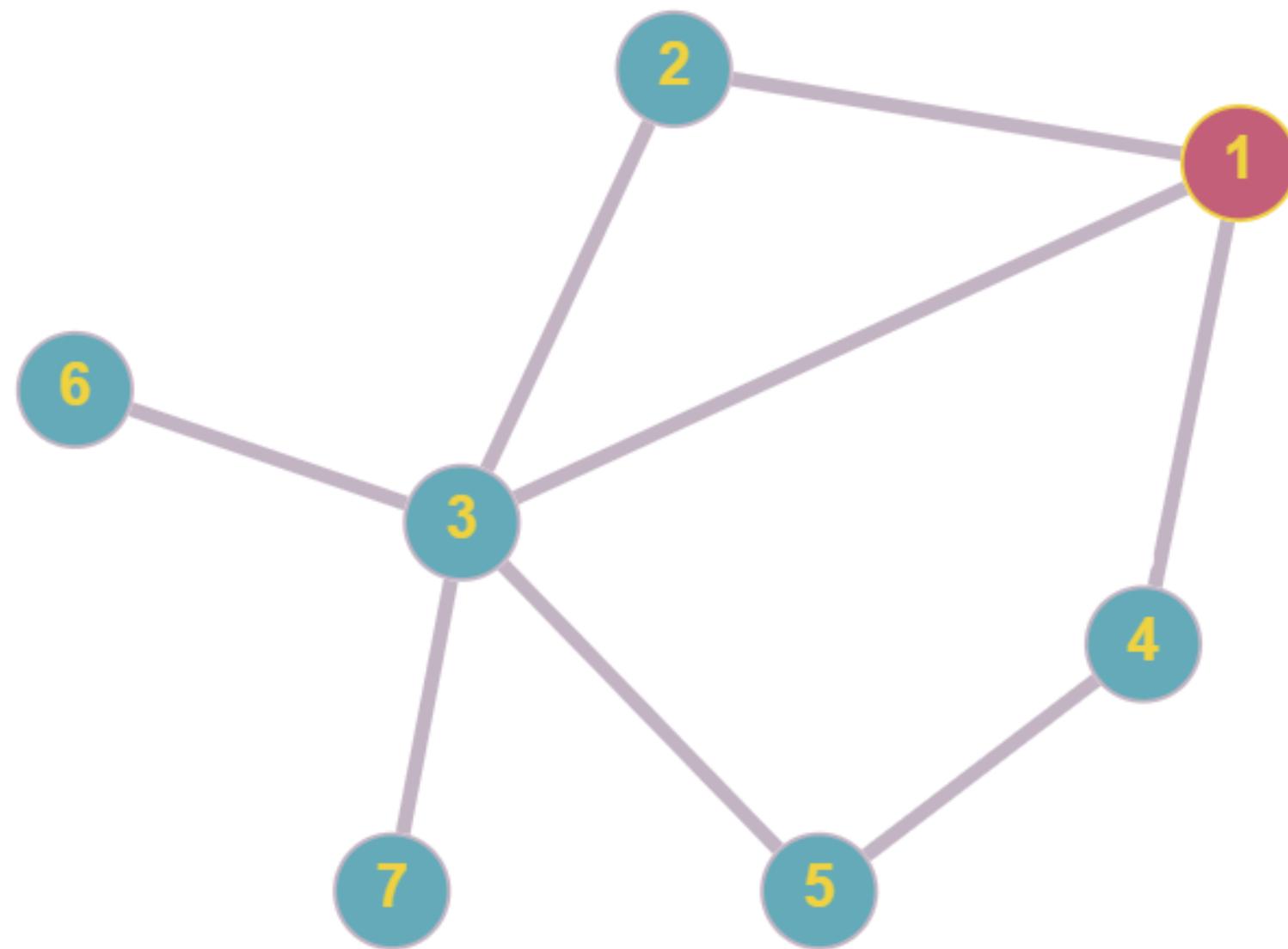


$G(V, A)$

$V: \{1, 2, 3, 4, 5, 6, 7\}$

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

Anchura



Paso 1: Inicialización

Comenzamos en el nodo 1.

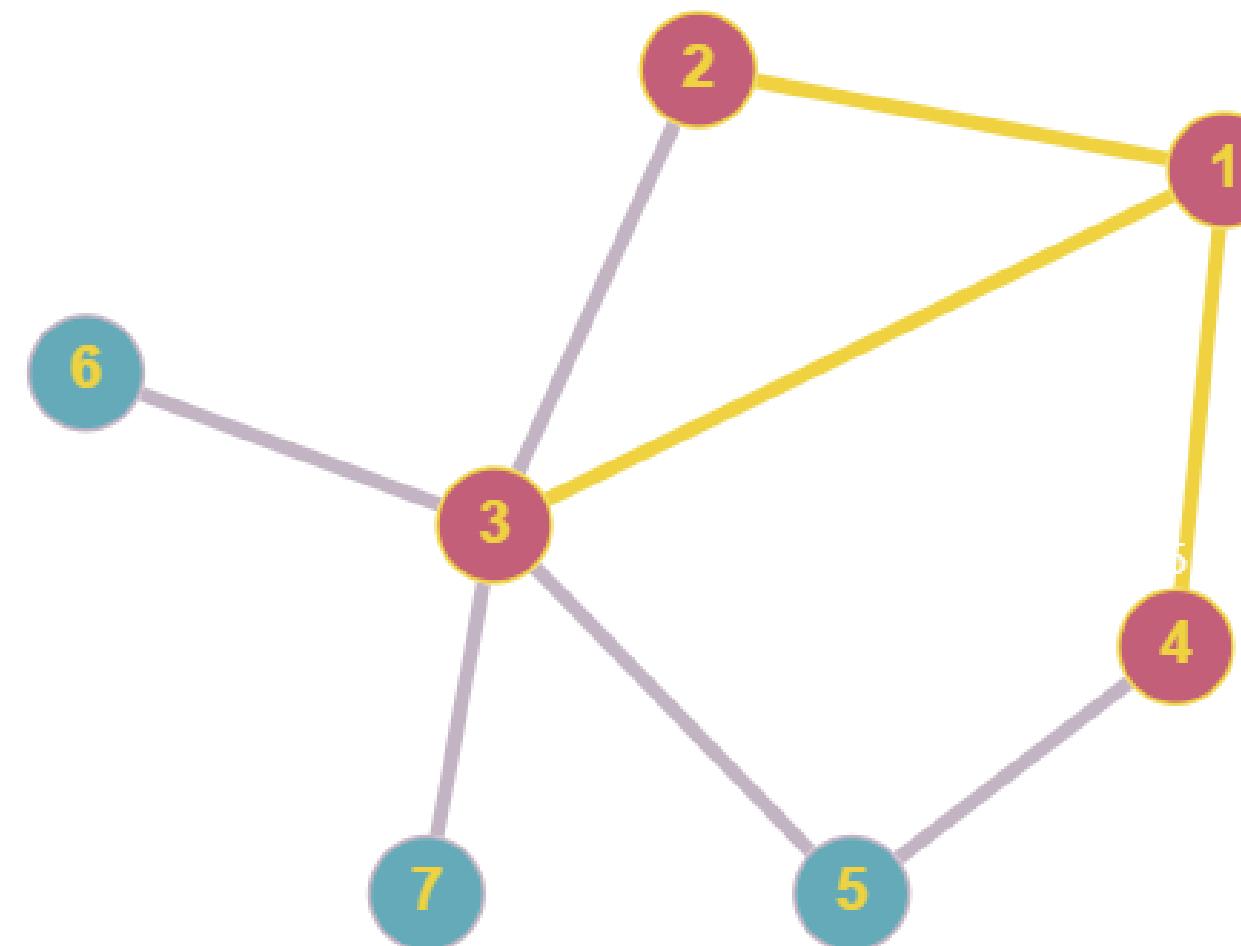
Se encola $1 \rightarrow$ Cola: [1]

Se marca 1 como visitado.

Lista de visitados:

1

Anchura



Paso 2: Procesamos el nodo 1

Se desencola 1 → Cola: []

Sus vecinos son 2, 3 y 4.

Se encolan en orden 2, 3 y 4 → Cola: [2, 3, 4]

Se marcan 2, 3 y 4 como visitados.

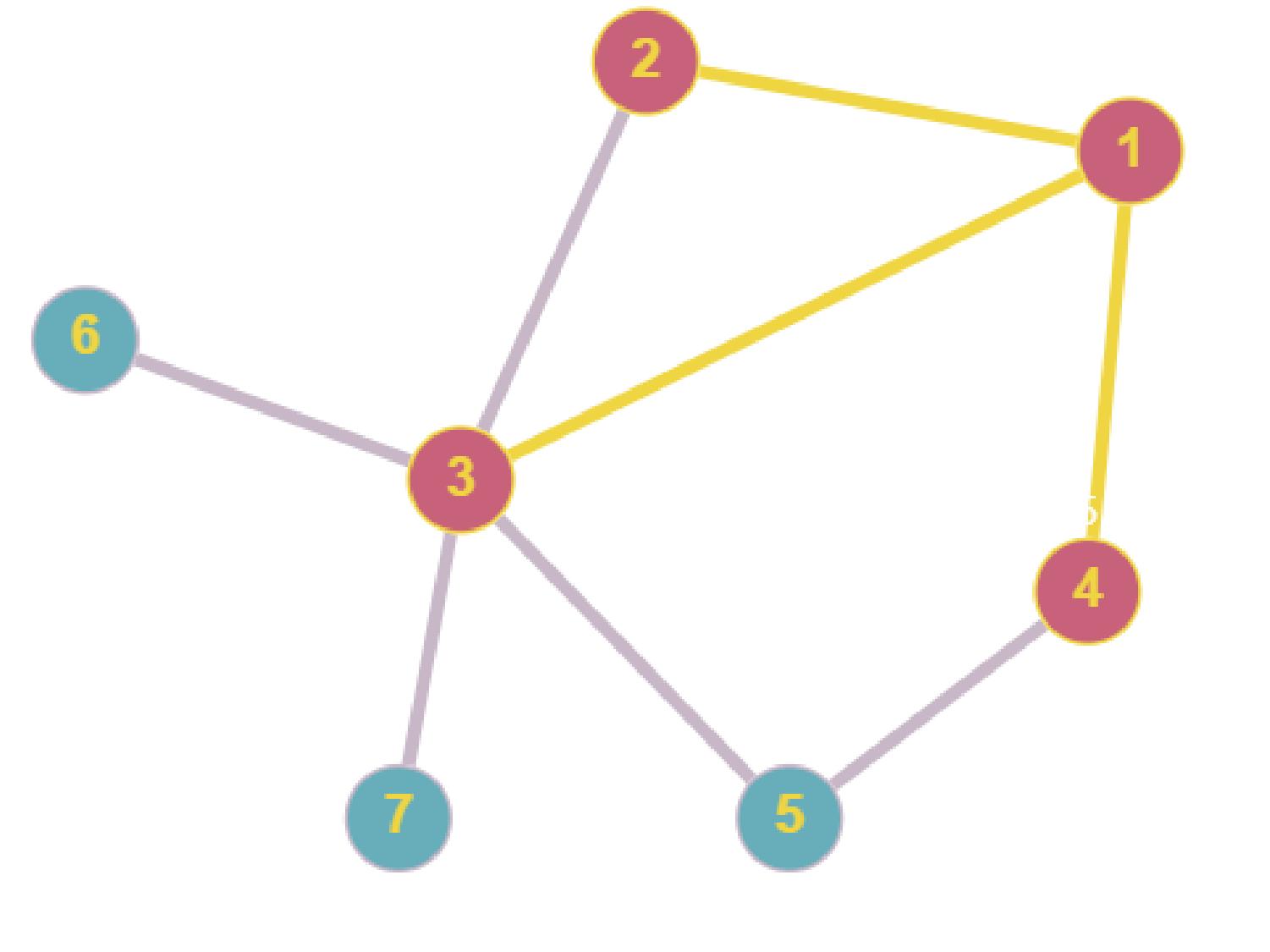
Lista de visitados:

1	2	3	4
---	---	---	---

Recorrido actual:

1

Anchura



Paso 3: Procesamos el nodo 2

- Se desencola 2 → Cola: [3, 4]
- No tiene vecinos que no hayan sido visitados.
- No se encola nada → Cola: [3, 4]

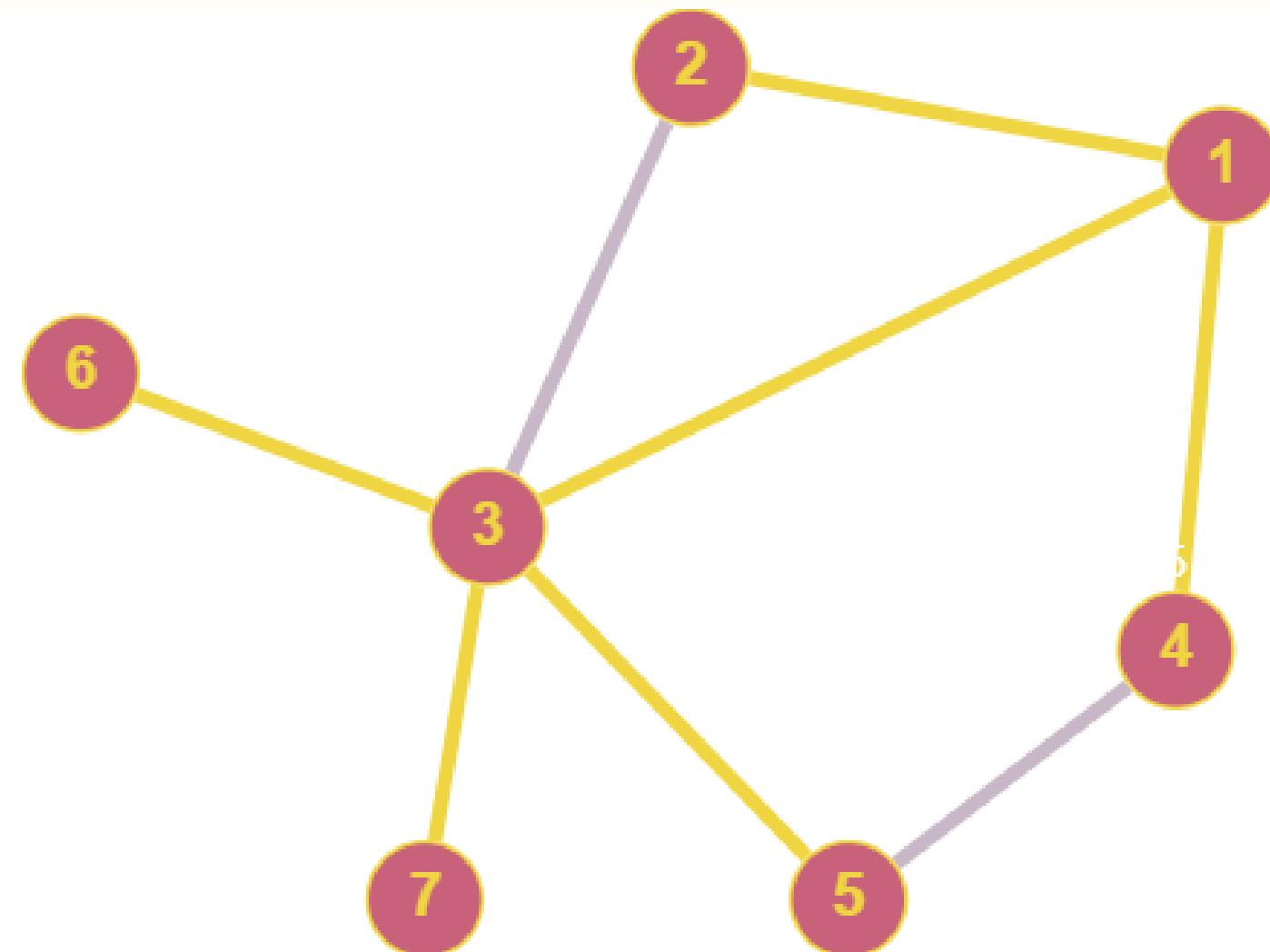
Lista de visitados:

1	2	3	4
---	---	---	---

Recorrido actual:

$1 \rightarrow 2$

Anchura



Paso 4: Procesamos el nodo 3

- Se desencola **3** → Cola: [4]
- Sus vecinos no visitados son 5, 6, 7.
- Se encolan 5, 6 y 7 → Cola: [4, 5, 6, 7]
- Se marcan 5, 6 y 7 como visitados.

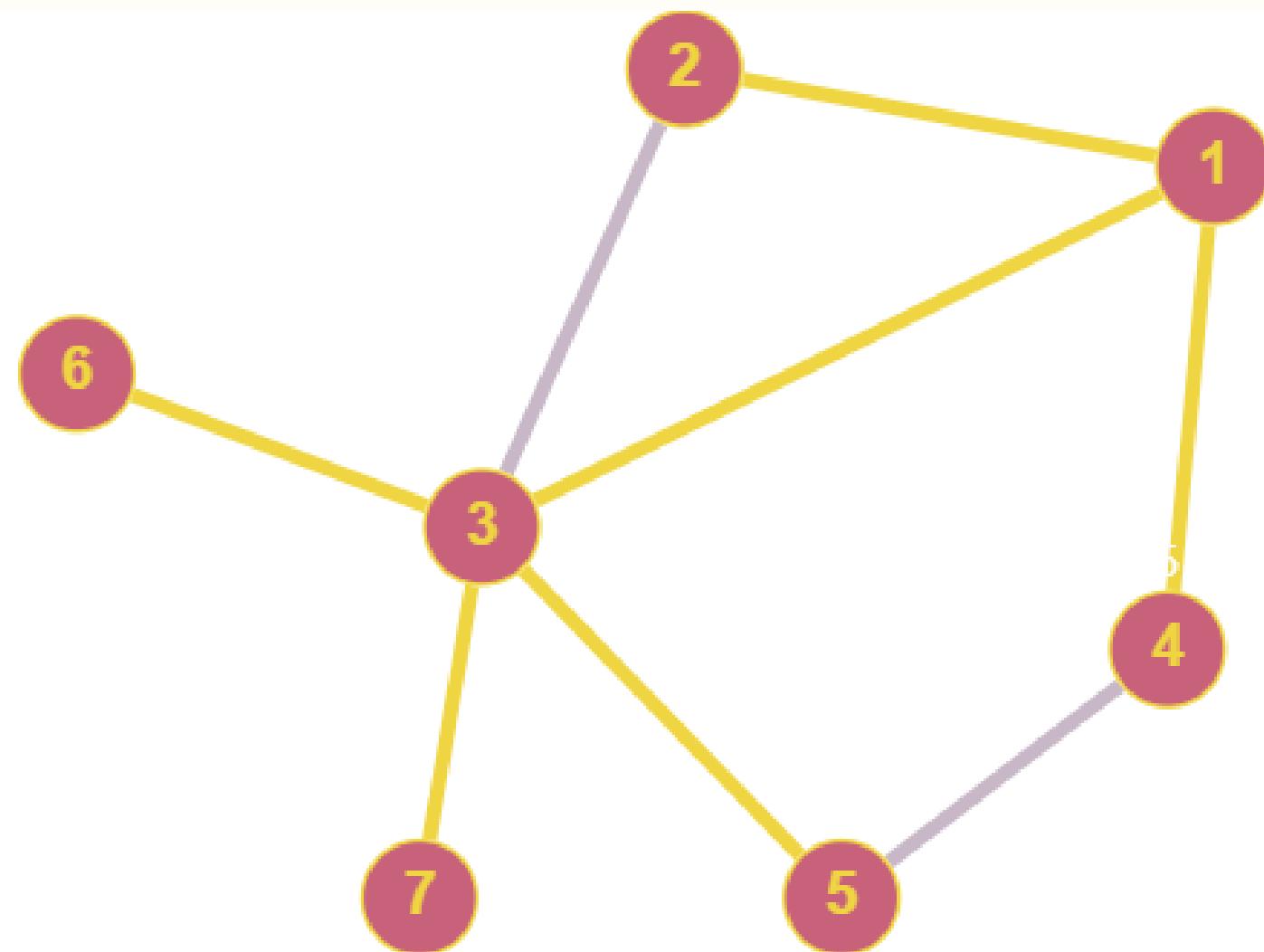
Lista de visitados:

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

Recorrido actual:

1 → 2 → 3

Anchura



Paso 5: Procesamos el nodo 4

- Se desencola 4 → Cola: [5, 6, 7]
- No tiene vecinos no visitados.
- No se encola nada nuevo.

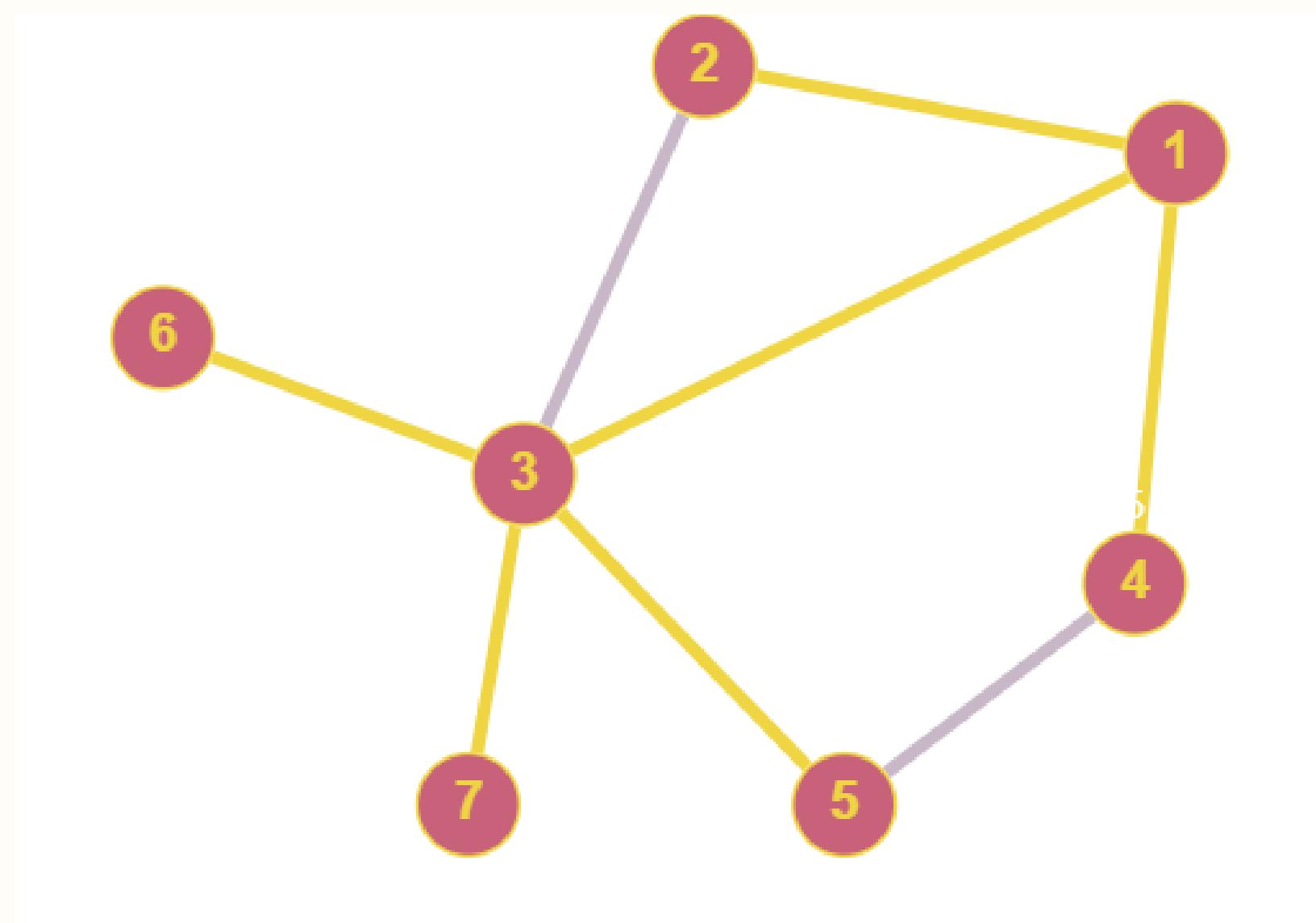
Lista de visitados:

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

Recorrido actual:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$

Anchura



Paso 6: Procesamos el nodo 5

- Se desencola $5 \rightarrow$ Cola: [6, 7]
- No tiene vecinos no visitados .

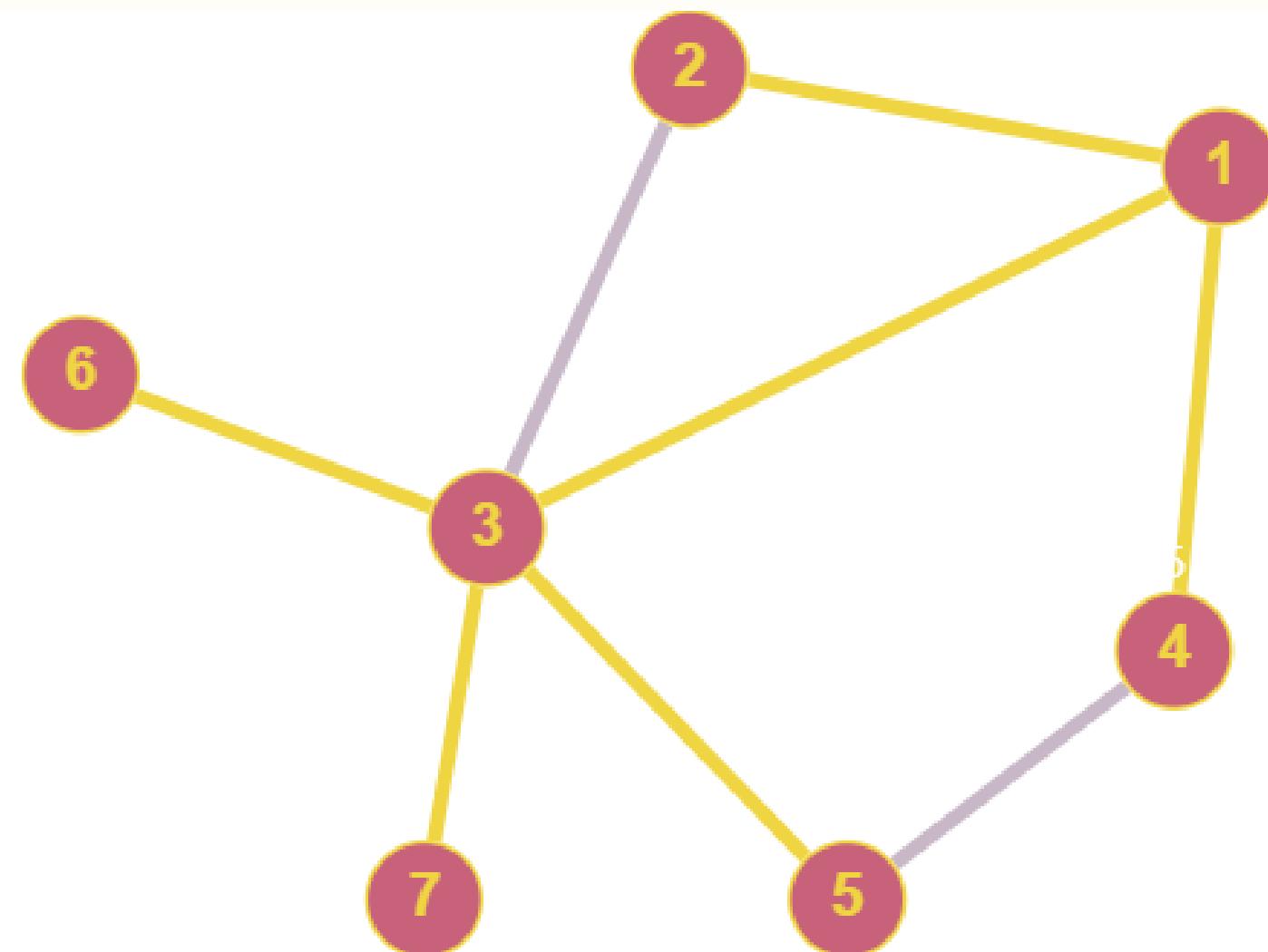
Lista de visitados:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Recorrido actual:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

Anchura



Paso 7: Procesamos el nodo 6

- Se desencola 6 → Cola: [7]
- No tiene vecinos no visitados.

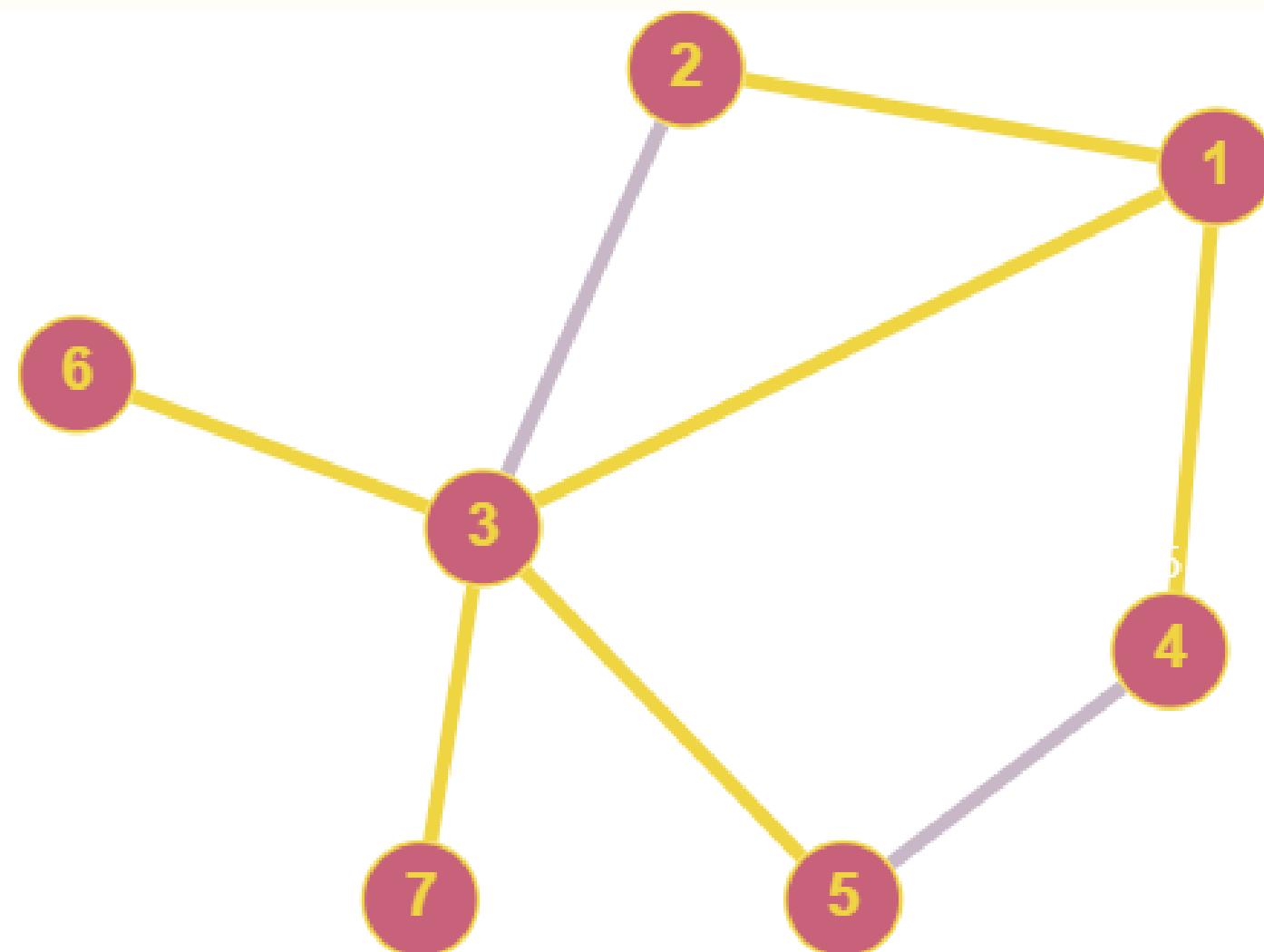
Lista de visitados:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Recorrido actual:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$

Anchura



Paso 8: Procesamos el nodo 7

- Se desencola $7 \rightarrow$ Cola: []
- No tiene vecinos no visitados.

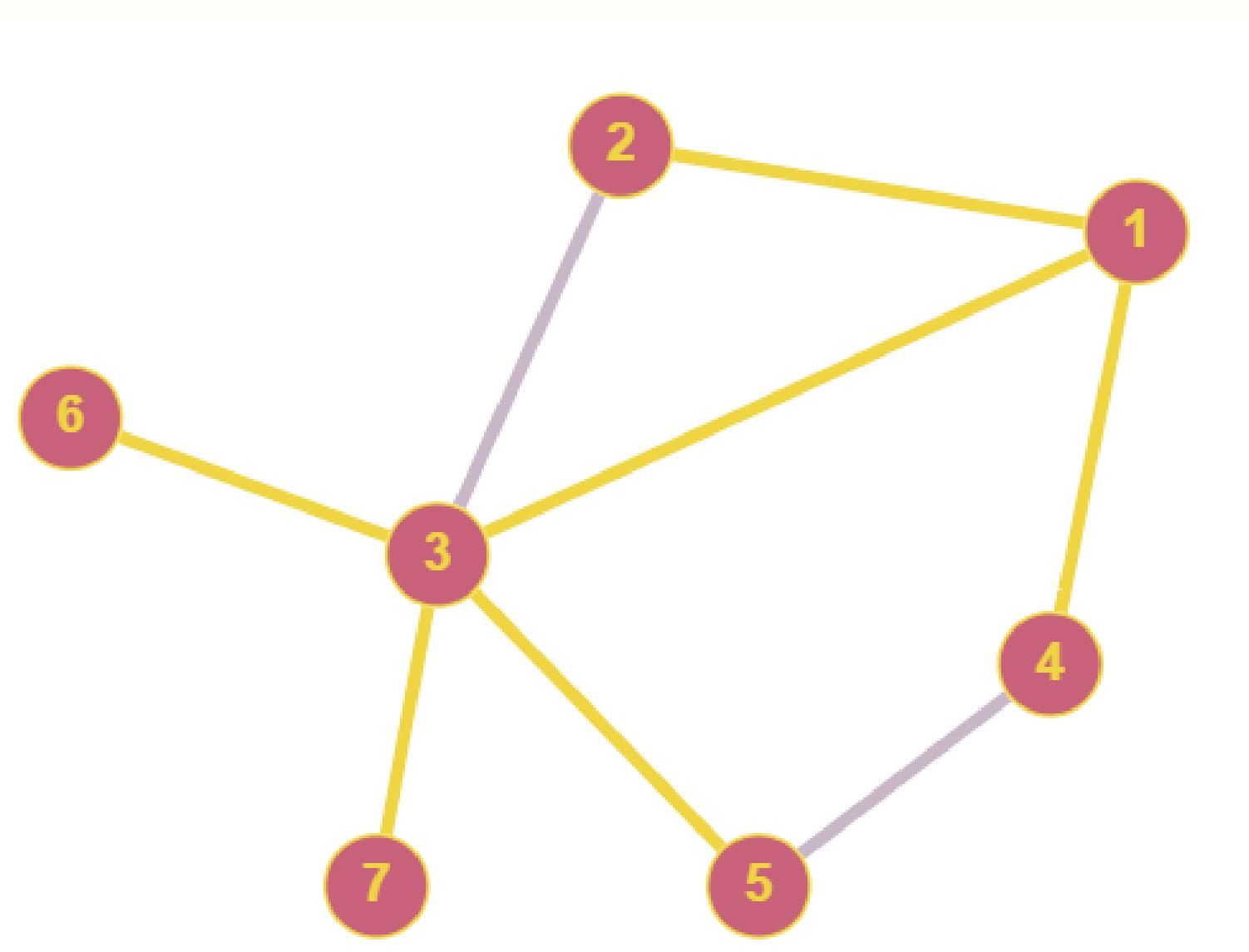
Lista de visitados:

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Recorrido actual:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$

Anchura



Recorrido final:

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7$

Lista de visitados final:

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

Anchura

Se desea realizar un recorrido por anchura comenzado desde el nodo 1 usando la siguiente matriz de adyacencia:

M	v1	v2	v3	v4
v1	0	1	0	1
v2	1	0	0	1
v3	0	0	0	1
v4	1	1	1	0

G(V,A)

V:{ 1, 2, 3, 4}

A:{(1,2), (1,4), (2,1), (2,4), (3,4), (4,1), (4,2), (4,3)}

Anchura

M	v1	v2	v3	v4
v1	0	1	0	1
v2	0	0	0	0
v3	0	0	0	0
v4	0	0	0	0

Paso 1: Inicialización
Comenzamos en el nodo 1.
Se encola 1 → Cola: [1]
Se marca 1 como visitado.

Lista de visitados:

1

M	v1
v1	0

Anchura

M	v1	v2	v3	v4
v1	0	1	0	1
v2	1	0	0	1
v3	0	0	0	1
v4	1	1	1	0

Paso 2: Procesamos el nodo 1

Se desencola 1 → Cola: []

Sus vecinos son 2 y 4.

Se encolan en orden 2 y 4 → Cola: [2, 4]

Se marcan 2 y 4 como visitados.

Lista de visitados:

1	2	4
---	---	---

M	v1	v2	v4
v1	0	1	1
v2	1	0	0
v4	1	0	0

Recorrido actual:

1

Anchura

M	v1	v2	v3	v4
v1	0	1	0	1
v2	1	0	0	1
v3	0	0	0	1
v4	1	1	1	0

M	v1	v2	v4
v1	0	1	1
v2	1	0	0
v4	1	0	0

Paso 3: Procesamos el nodo 2

- Se desencola $2 \rightarrow$ Cola: [4]
- No tiene vecinos que no hayan sido visitados.
- No se encola nada \rightarrow Cola: [4]

Lista de visitados:

1	2	4
---	---	---

Recorrido actual:

$1 \rightarrow 2$

Anchura

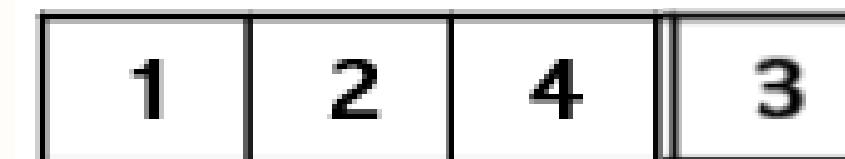
M	v1	v2	v3	v4
v1	0	1	0	1
v2	1	0	0	1
v3	0	0	0	1
v4	1	1	1	0

Paso 4: Procesamos el nodo 4

- Se desencola 4 → Cola: []
- Tiene como vecino no visitado el 3.
- Se encola 3 → Cola: [3].
- Se marca el 3 como visitado

Lista de visitados:

M	v1	v2	v4	v3
v1	0	1	1	0
v2	1	0	0	0
v4	1	0	0	1
v3	0	0	1	0



Recorrido actual:

$1 \rightarrow 2 \rightarrow 4$

Anchura

M	v1	v2	v3	v4
v1	0	1	0	1
v2	1	0	0	1
v3	0	0	0	1
v4	1	1	1	0

Paso 5: Procesamos el nodo 3

- Se desencola $3 \rightarrow$ Cola: []
- No tiene vecinos no visitados.
- No se encola nada nuevo.

Lista de visitados:

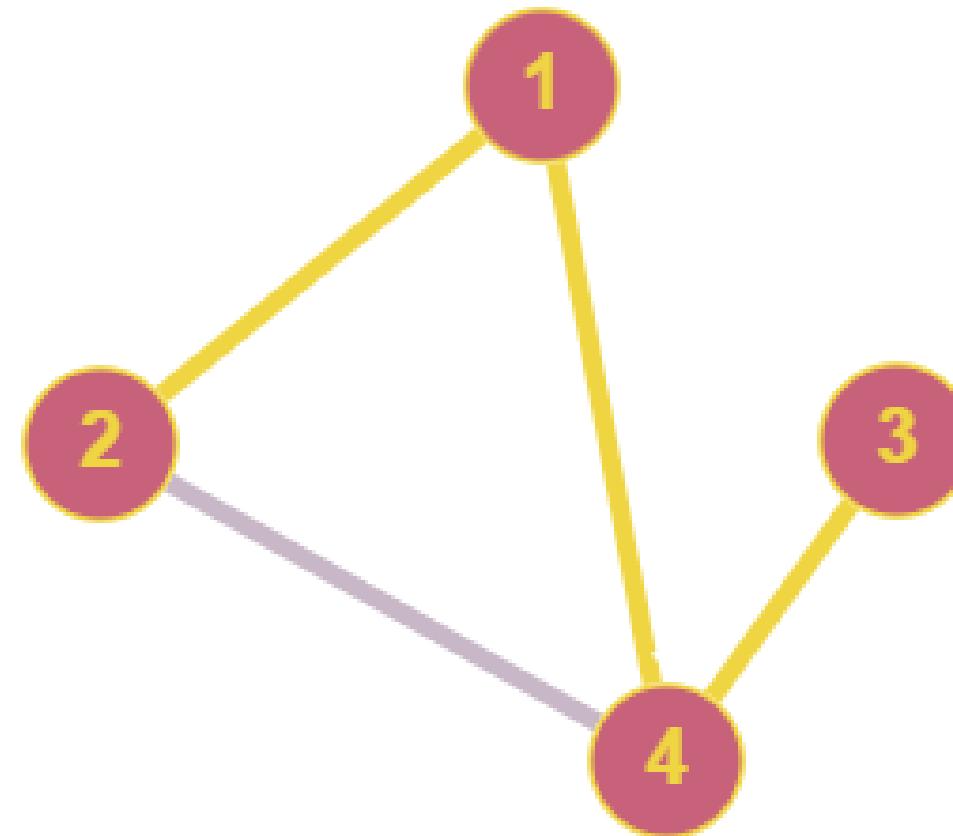


M	v1	v2	v4	v3
v1	0	1	1	0
v2	1	0	0	0
v4	1	0	0	1
v3	0	0	1	0

Recorrido actual:

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3$

Anchura



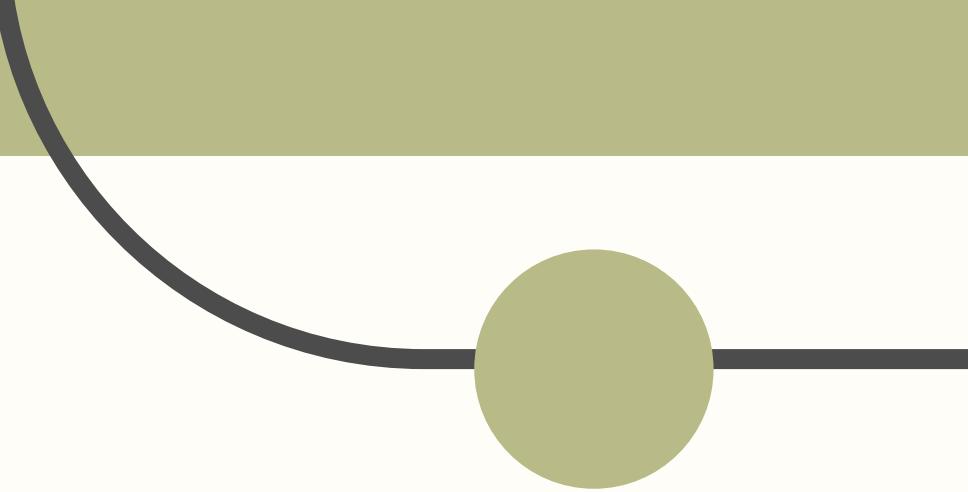
Recorrido final:

$1 \rightarrow 2 \rightarrow 4 \rightarrow 3$

Lista de visitados final:

1	2	4	3
---	---	---	---

M	V1	V2	V4	V3
V1	0	1	1	0
V2	1	0	0	0
V4	1	0	0	1
V3	0	0	1	0



→ PROFUNDIDAD

Una búsqueda en profundidad es un algoritmo de búsqueda para lo cual recorre los nodos de un grafo. Su funcionamiento consiste en ir expandiendo cada uno de los nodos que va localizando, de forma recurrente (desde el nodo padre hacia el nodo hijo). Cuando ya no quedan más nodos que visitar en dicho camino, regresa al nodo predecesor, de modo que repite el mismo proceso con cada uno de los vecinos del nodo.

DFS: Depth-First Search

Recorrido por profundidad

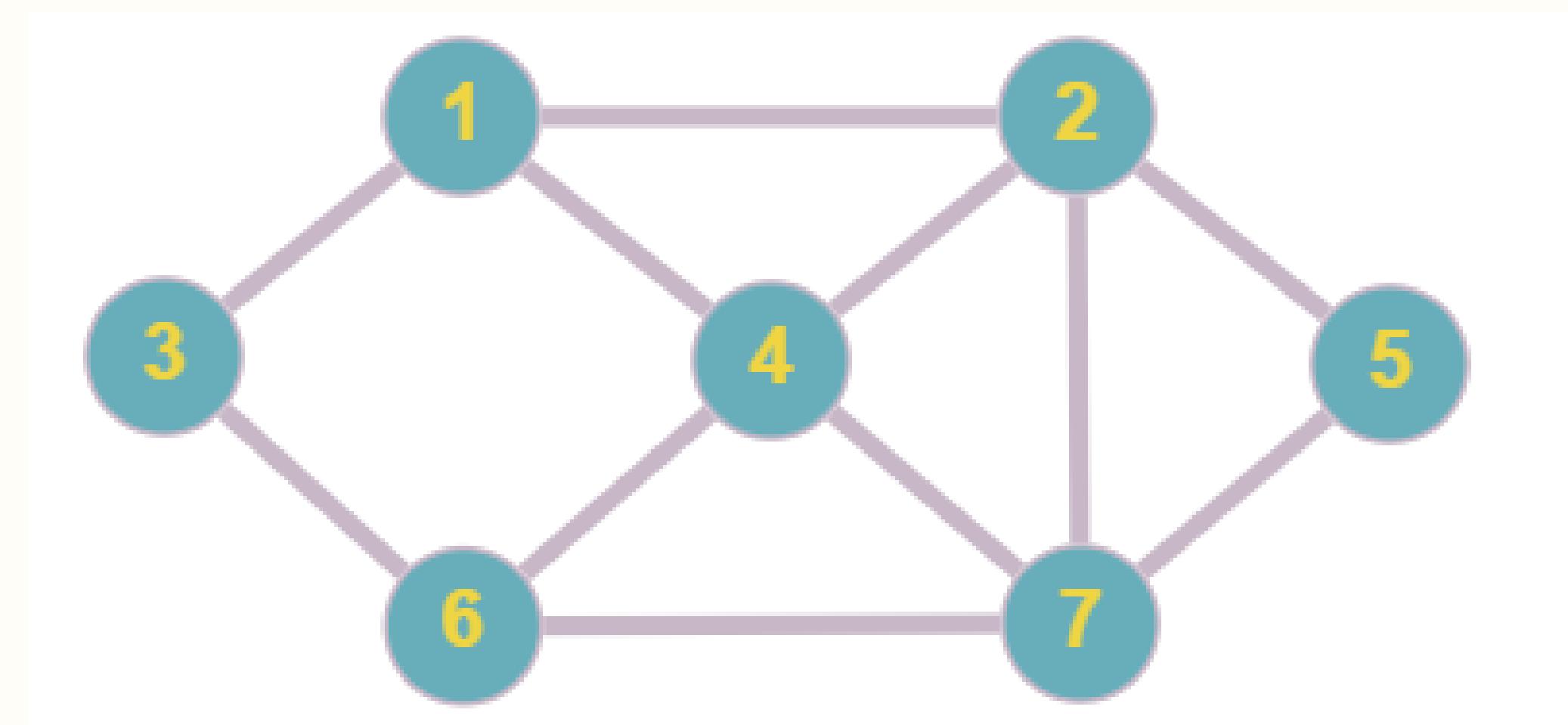
➔ PROFUNDIDAD

- Comienza de un nodo elegido arbitrariamente llamado nodo raíz
- Se pone el nodo en la pila y en la lista
- Se toma al primer vecino en orden que no esté en la lista como nodo raíz
- De no haber más vecinos o todos están dentro de la lista, se saca un elemento de la pila como nuevo nodo raíz
- Termina cuando no hay elementos en la pila al momento de sacar

Utilizado detectar ciclos de manera eficiente

Profundidad

Se desea realizar un recorrido por profundidad del siguiente grafo

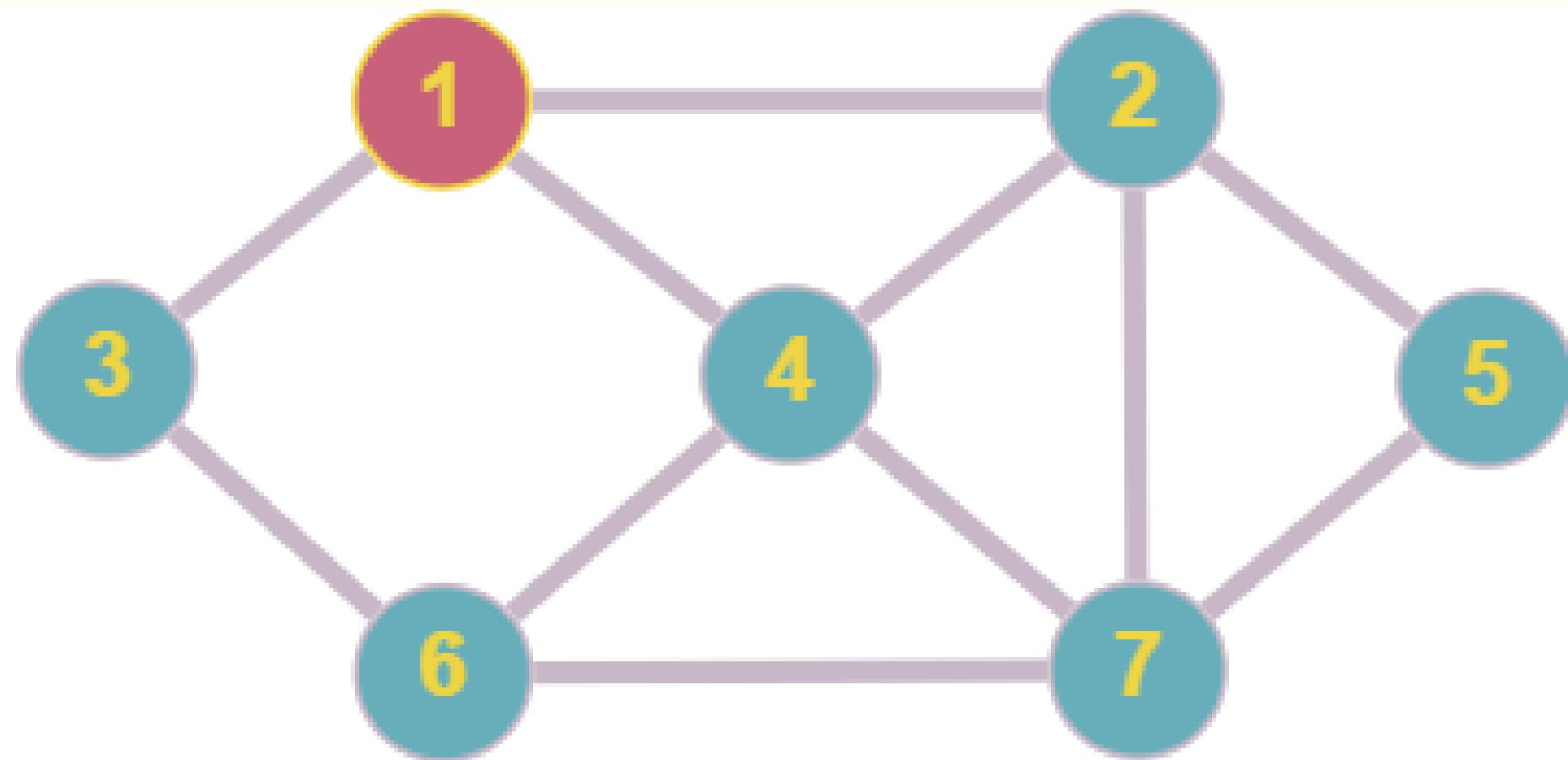


G(V,A)

V:{ 1, 2, 3, 4, 5, 6, 7}

A:{(1,2), (1,3), (1,4), (2,1), (3,1), (4,1), (2,4), (2,7), (2,5), (4,2), (7,2), (5,2), (5,7), (7,5), (7,4), (7,6), (4,7), (6,7), (6,4), (6,3), (4,6), (3,6)}

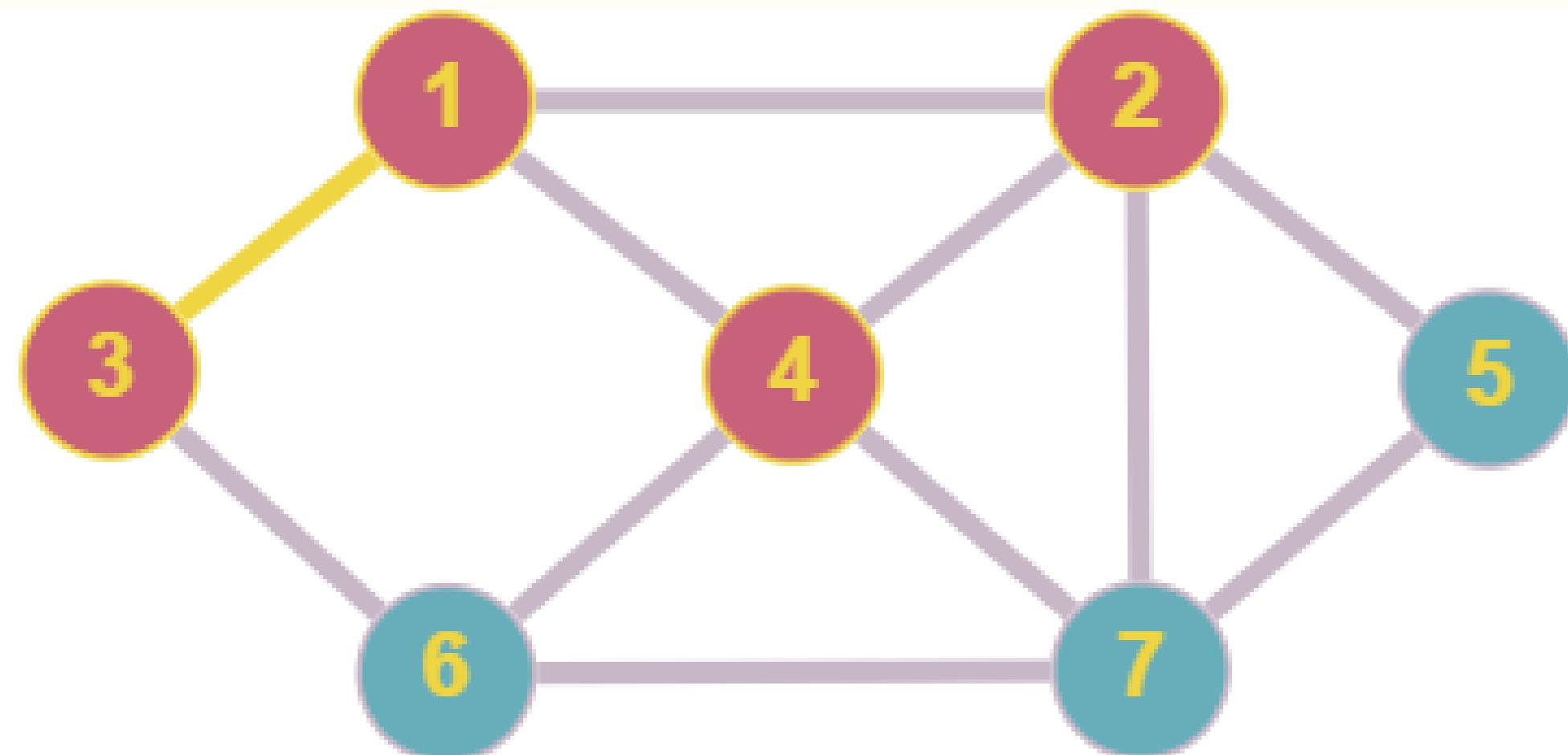
Profundidad



Inicialización:

- Elegimos un nodo inicial, por ejemplo, el nodo 1.
- Creamos una pila donde almacenaremos los nodos pendientes por visitar.
- Creamos una lista para registrar los nodos ya visitados.
- Pila inicial: [1]
- Visitados: [1]

Profundidad



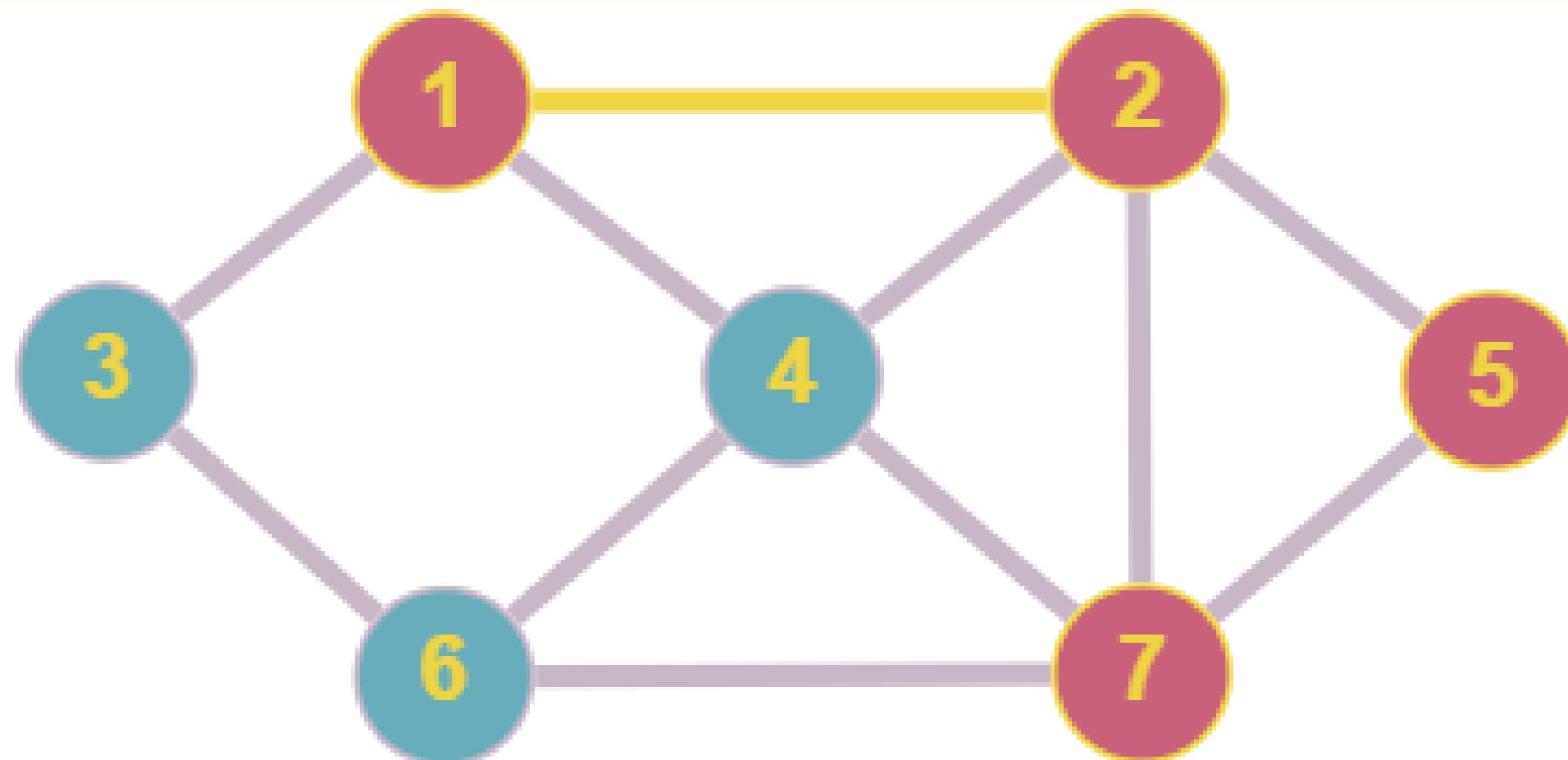
Paso 1: Visitar el nodo en la cima de la pila.

- Sacamos el nodo 1 de la pila (lo estamos visitando).
- Revisamos los nodos conectados al nodo 1 que no han sido visitados.
- Agregamos esos nodos a la pila en orden (por ejemplo, 3, 4 y 2).
- Pila: [3, 4, 2]
- Visitados: [1, 2, 3, 4]

Recorrido actual:

1

Profundidad



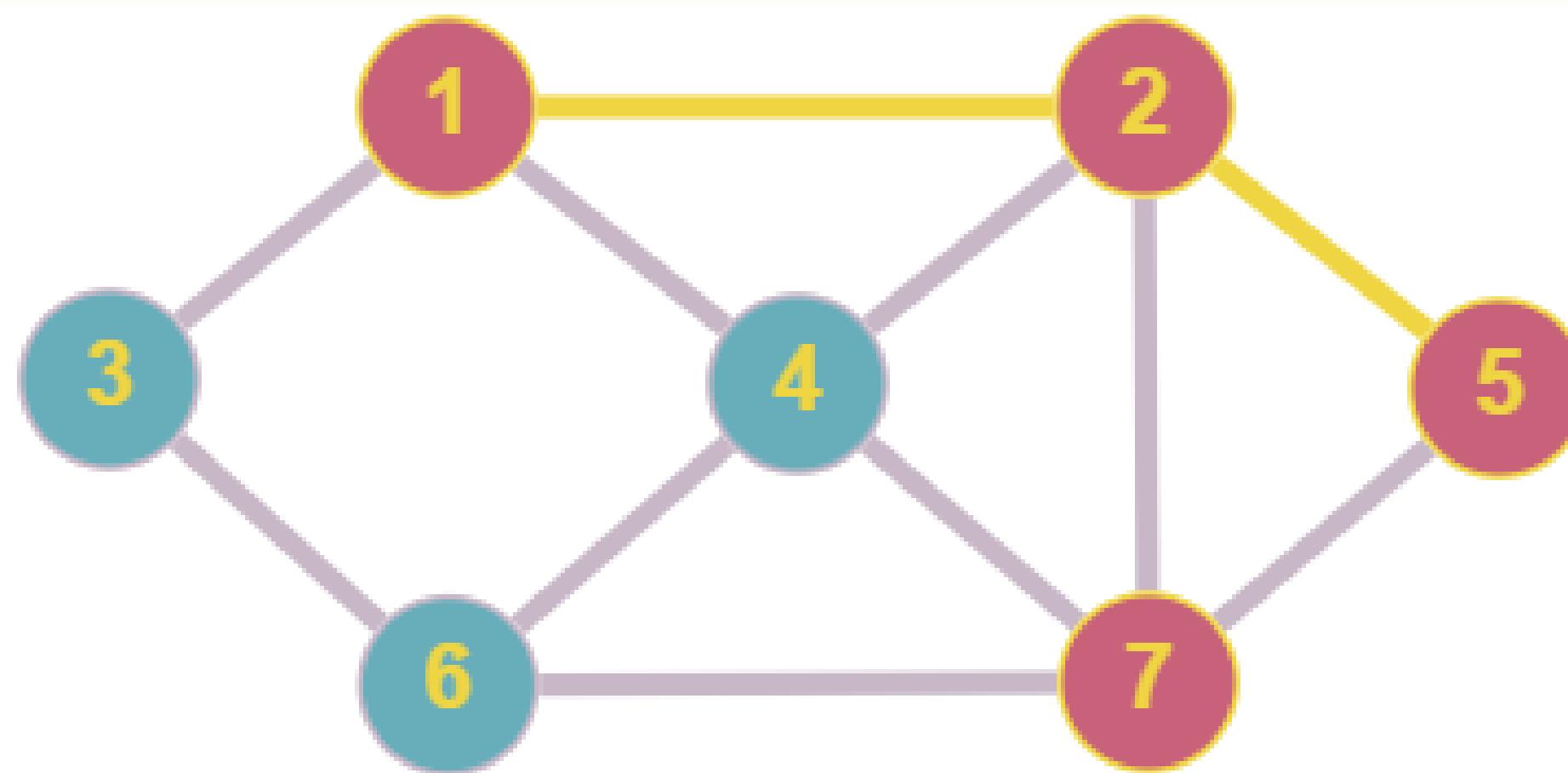
Paso 2: Procesamos el nodo 2

1. Extraer de la pila: Se retira el nodo 2 (ubicado en la cima).
2. Agregar vecinos de 2:
 - o Vecinos sin visitar de 2: 5 y 7.
3. Estado de la pila: [3, 4, 7, 5]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7]

Recorrido actual:

$1 \rightarrow 2$

Profundidad



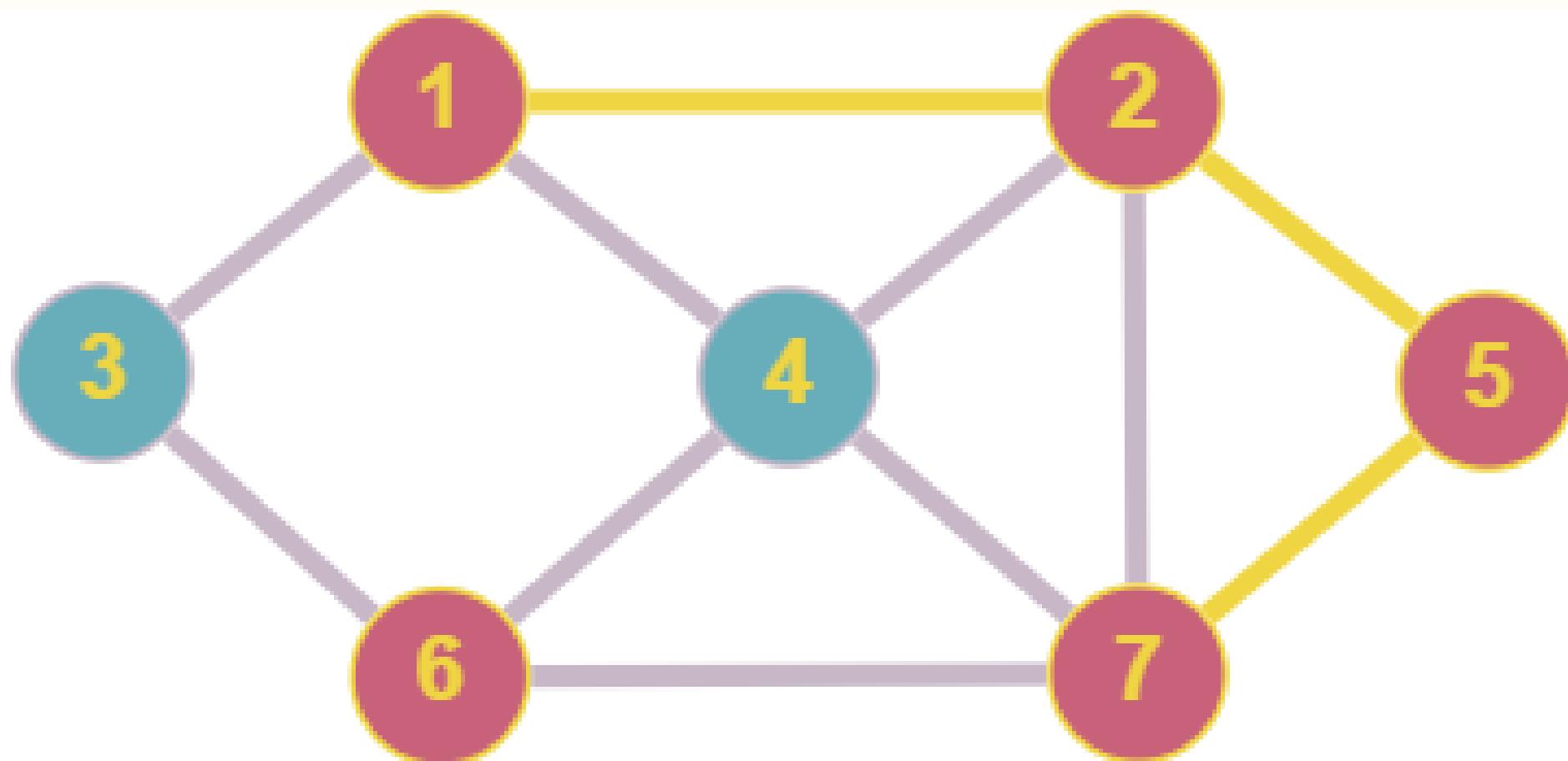
Paso 3: Procesamos el nodo 5

1. Extraer de la pila: Se retira el nodo 5 (ubicado en la cima).
2. Agregar vecinos de 5:
 - o Vecinos sin visitar de : (vacío).
3. Estado de la pila: [3, 4, 7]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7]

Recorrido actual:

$1 \rightarrow 2 \rightarrow 5$

Profundidad



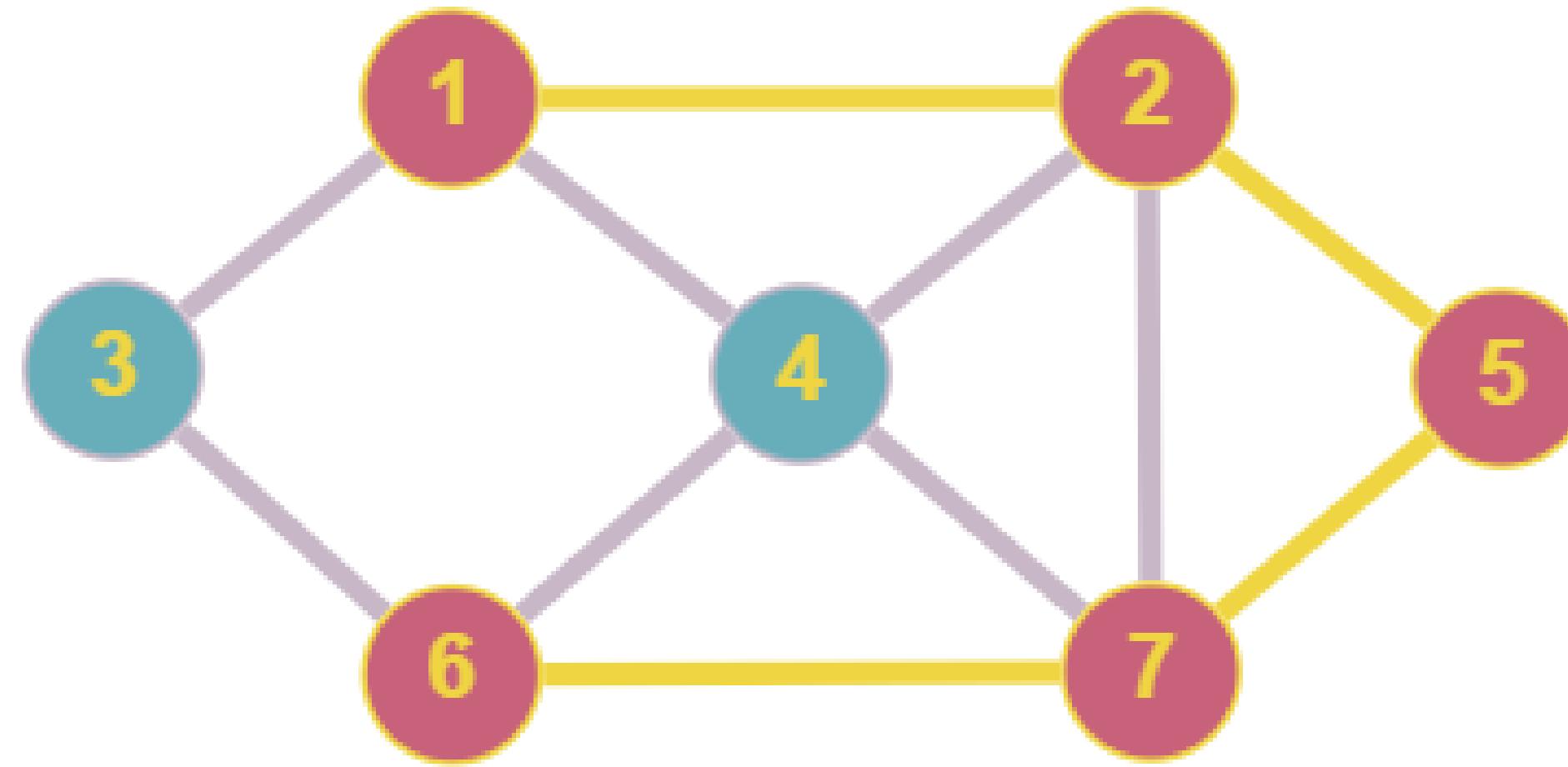
Paso 4: Procesamos el nodo 7

1. Extraer de la pila: Se retira el nodo 7 (ubicado en la cima).
2. Agregar vecinos de 7:
 - o Vecinos sin visitar de 7: 6.
3. Estado de la pila: [3, 4, 6]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7, 6]

Recorrido actual:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 7$

Profundidad



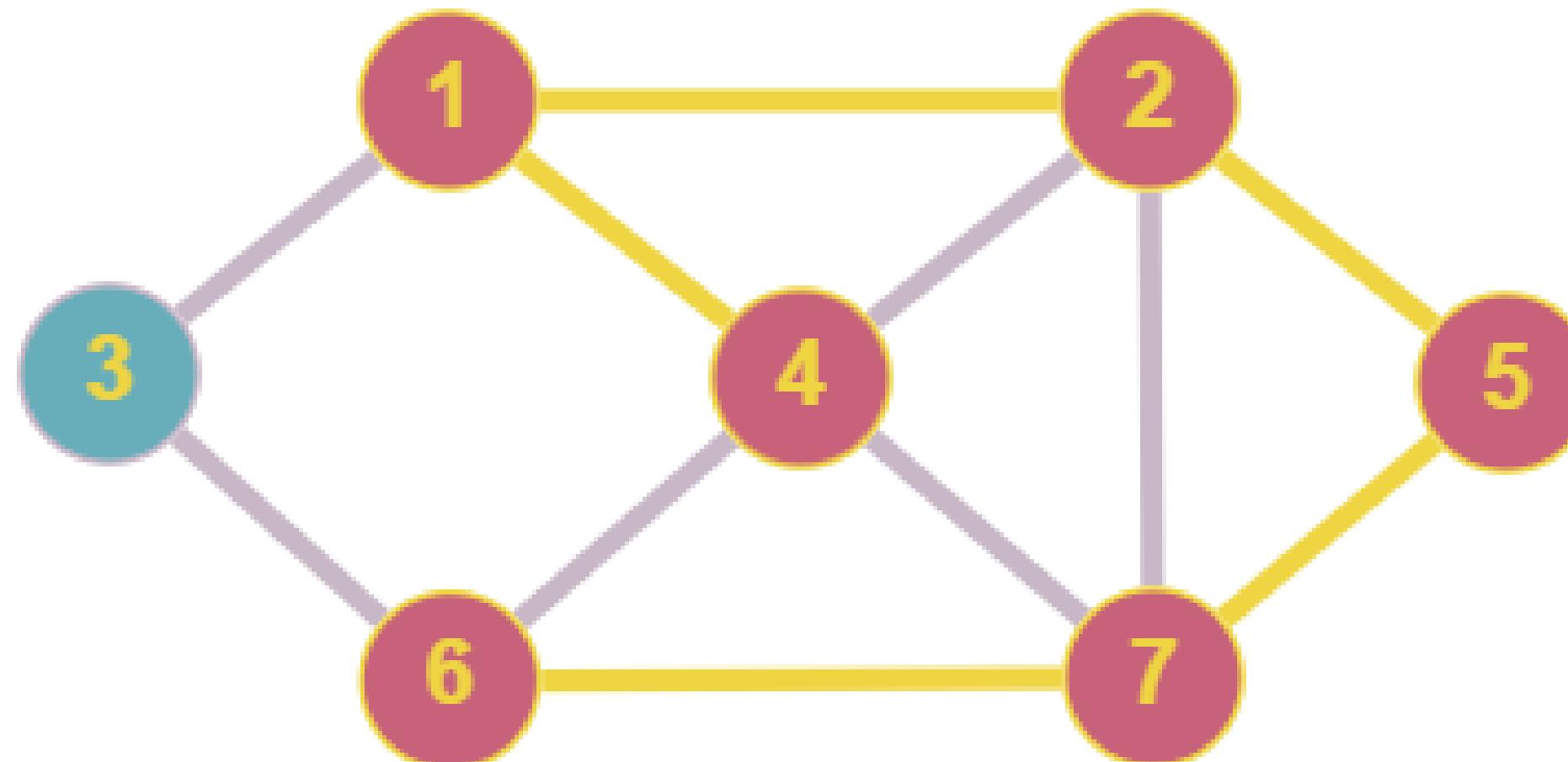
Paso 5: Procesamos el nodo 6

1. Extraer de la pila: Se retira el nodo 6 (ubicado en la cima).
2. Agregar vecinos de 6:
 - o Vecinos sin visitar de 6: (vacío).
3. Estado de la pila: [3, 4]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7, 6]

Recorrido actual:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 6$

Profundidad



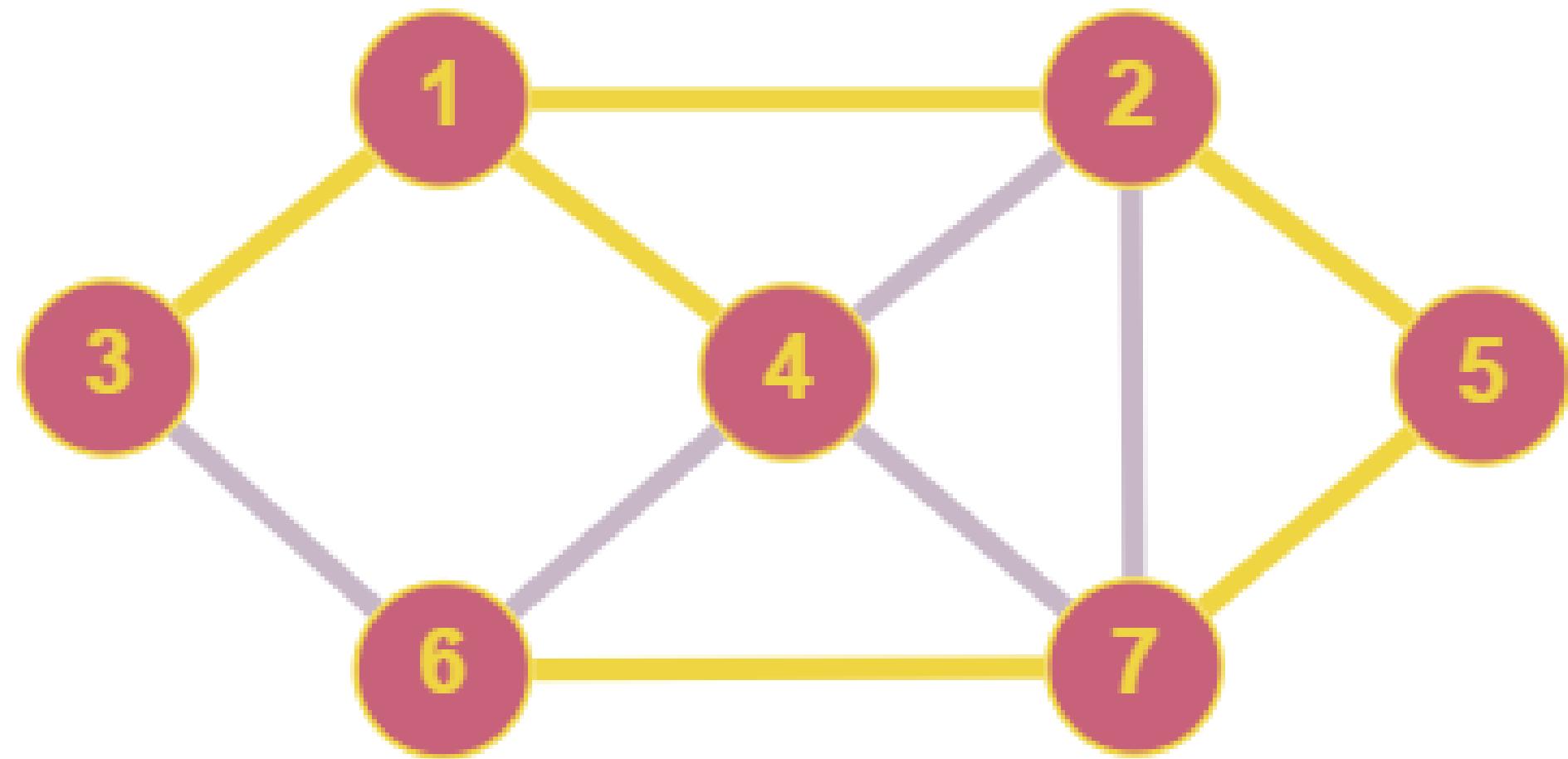
Paso 6: Procesamos el nodo 4

1. Extraer de la pila: Se retira el nodo 4 (ubicado en la cima).
2. Agregar vecinos de 4:
 - o Vecinos sin visitar de 4: (vacío).
3. Estado de la pila: [3]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7, 6]

Recorrido actual:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 4$

Profundidad



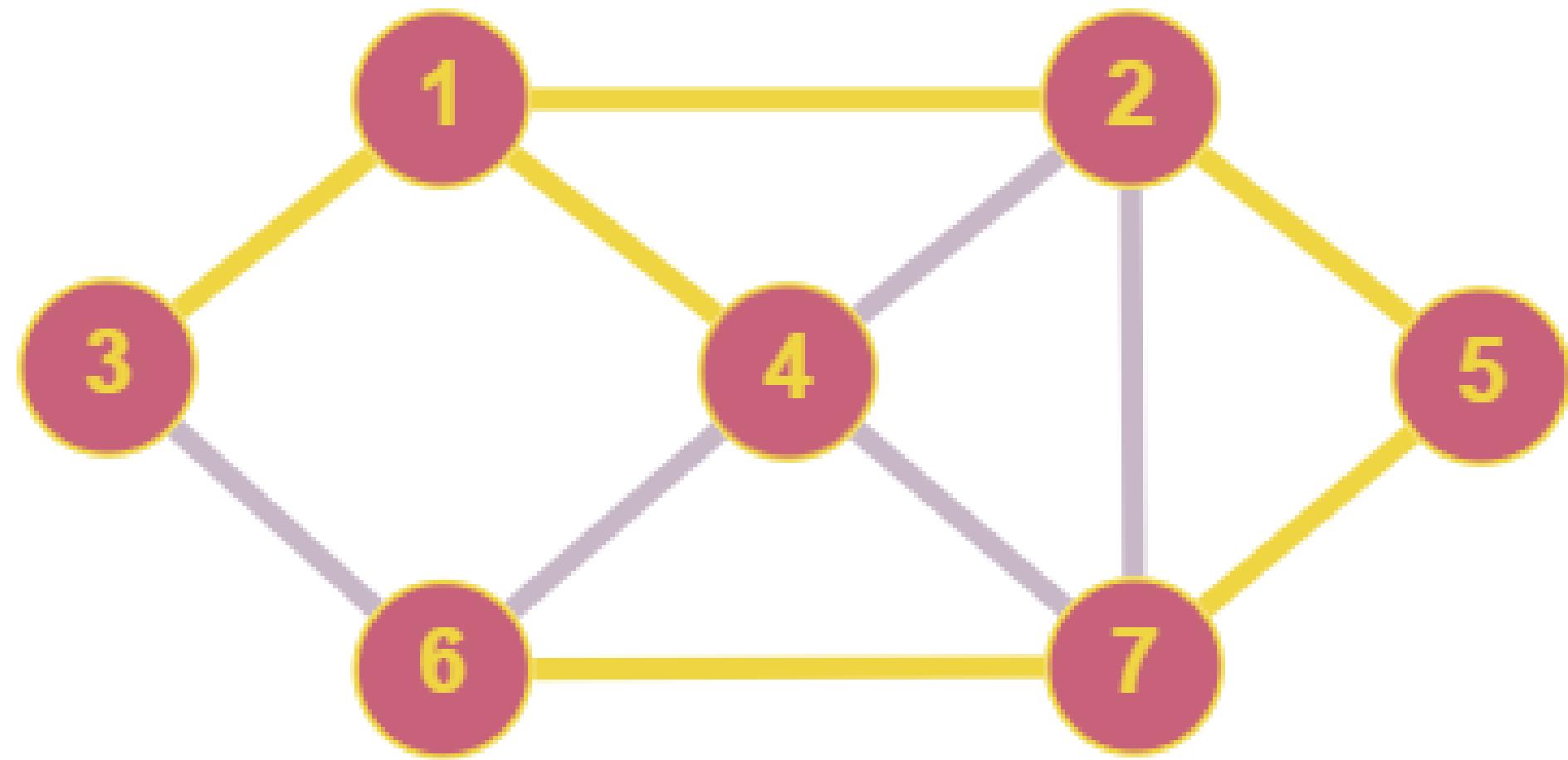
Paso 7: Procesamos el nodo 3

1. Extraer de la pila: Se retira el nodo 3 (ubicado en la cima).
2. Agregar vecinos de 3:
 - o Vecinos sin visitar de 3: (vacío).
3. Estado de la pila: [vacío]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7, 6]

Recorrido final:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow 3$

Profundidad



Paso 7: Procesamos el nodo 3

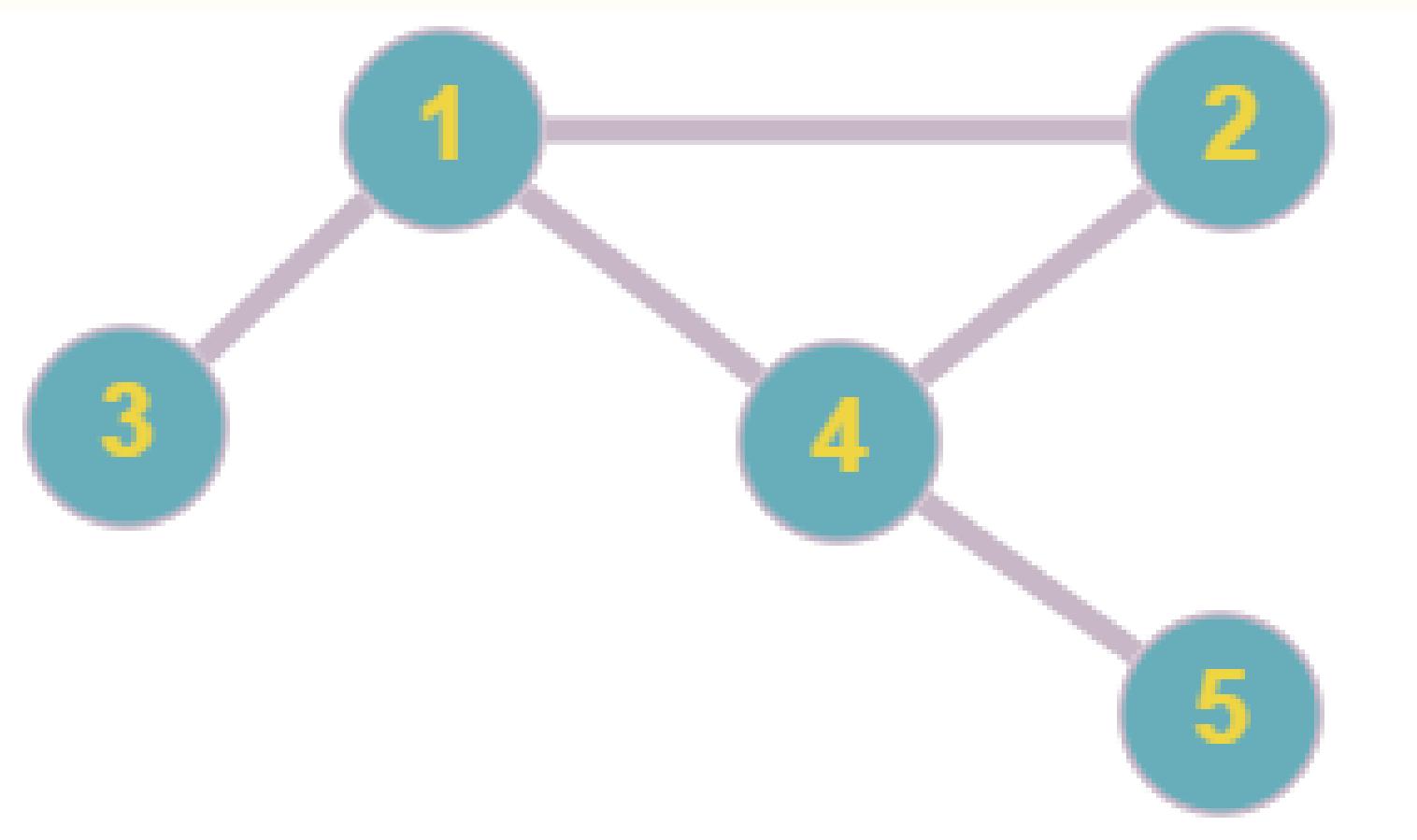
1. Extraer de la pila: Se retira el nodo 3 (ubicado en la cima).
2. Agregar vecinos de 3:
 - o Vecinos sin visitar de 3: (vacío).
3. Estado de la pila: [vacío]
4. Marcar como visitado:
 - a. Visitados: [1, 2, 3, 4, 5, 7, 6]

Recorrido final:

$1 \rightarrow 2 \rightarrow 5 \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow 3$

Profundidad

Se desea realizar un recorrido por profundidad del siguiente grafo por medio de una lista de adyacencia



Lista de adyacencia

1 → (2) → (3) → (4)
2 → (1) → (4)
3 → (1)
4 → (1) → (2) → (5)
5 → (4)

G(V,A)

V:{ 1, 2, 3, 4, 5}

A:{(1,2), (1,3), (2,1), (3,1), (2,4), (4,2), (5,4), (4,5), (1,4), (4,1)}

Profundidad

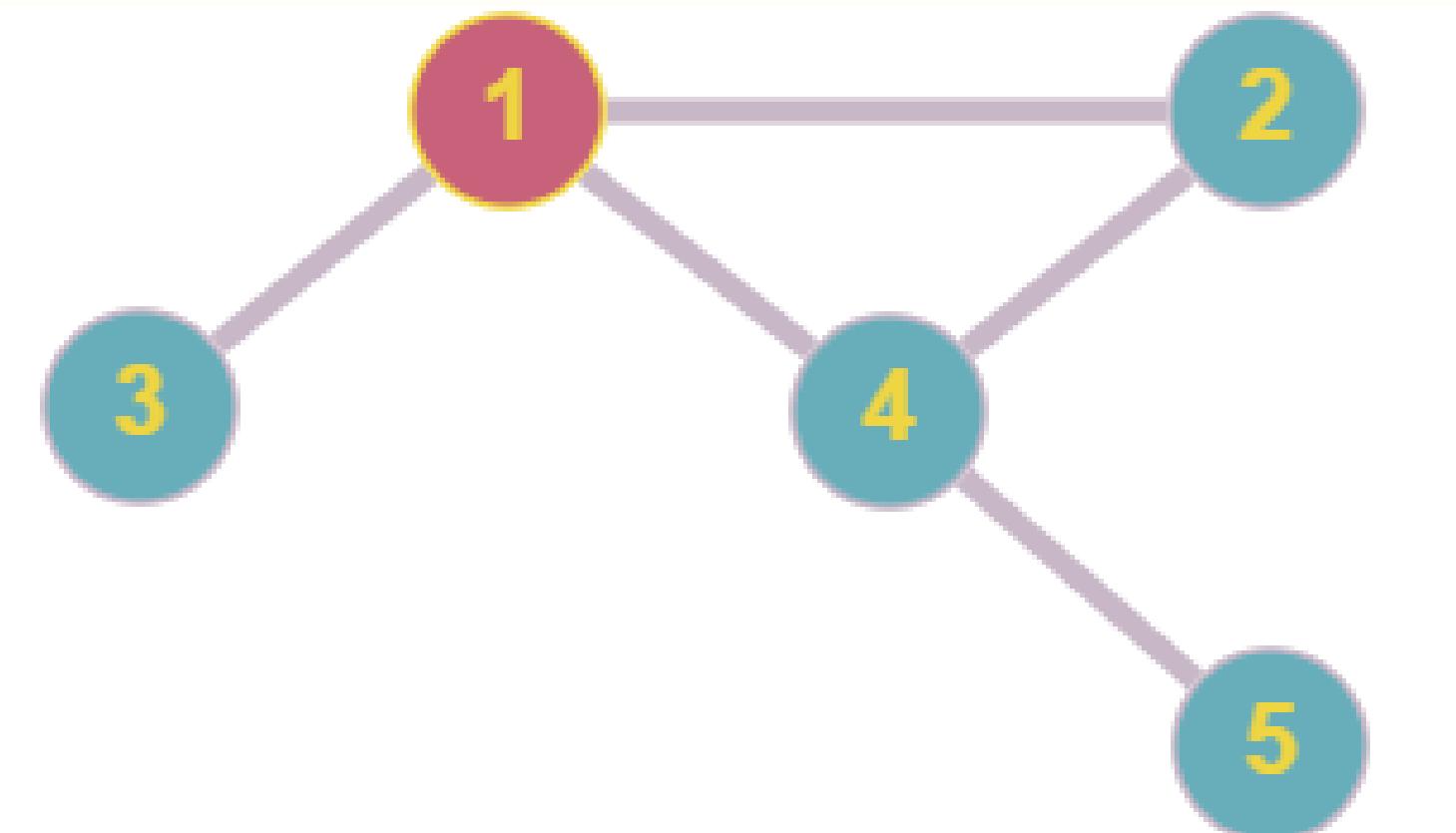
Inicializamos la pila, (escogemos un nodo raiz lo marcamos como visitado y lo ponemos en la pila)

1. Expulsamos el nodo en el tope de la pila.

2. Añadimos sus vecinos no visitados a la lista de visitados.

3. Se apilan los nodos vecinos que acabamos de añadir a la lista de visitados.

4. Repetir hasta que ya no hayan más nodos que visitar.



1 → (2) → (3) → (4)

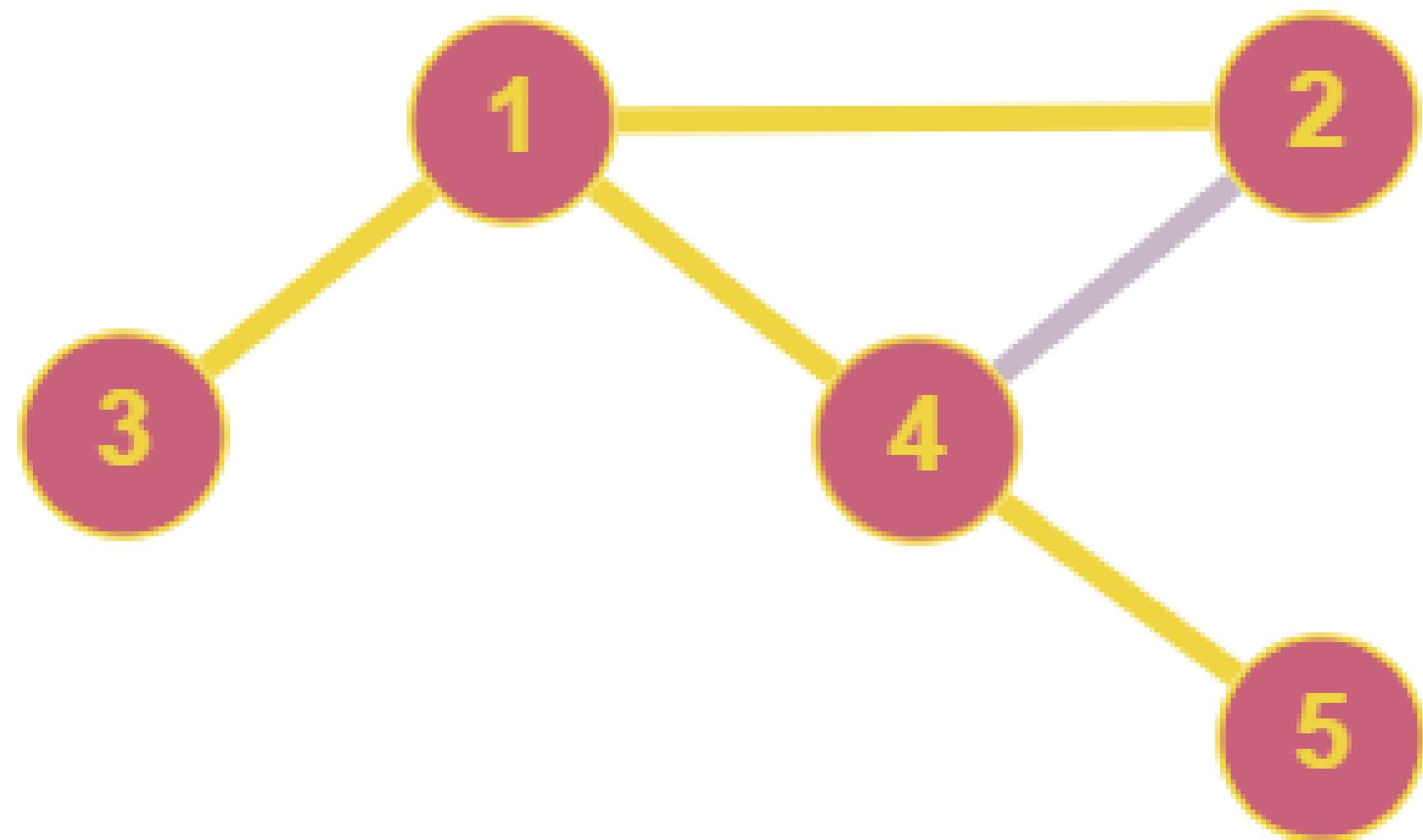
2 → (1) → (4)

3 → (1)

4 → (1) → (2) → (5)

5 → (4)

Profundidad



recorrido final en DFS:
 $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$

EJERCICIOS

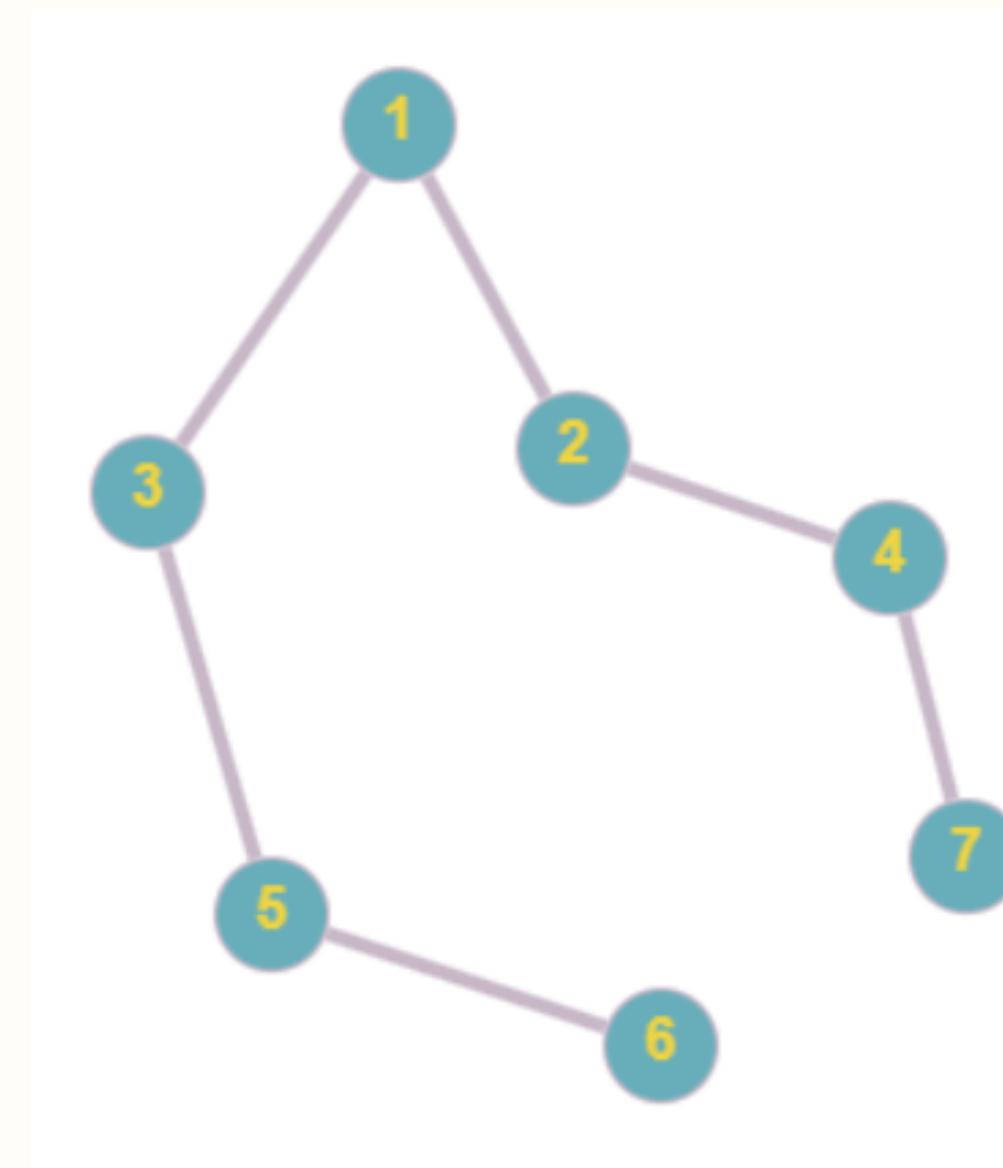
Ejercicio

Realizar un recorrido en anchura iniciando desde el nodo v1 del siguiente grafo:

G(V,A)

V:{ 1, 2, 3, 4, 5, 6, 7}

A:{(1,2), (1,3), (2,1), (3,1), (3,5), (2,4),
(4,2), (5,3), (5,6), (6,5),(6,7), (7,6),
(4,6), (6,4)}



Solución

Árbol de expansión:

Recorrido final:

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_7 \rightarrow v_6$

Inicio en $v_1 \rightarrow$ Cola: $[v_1]$

Procesamos: $v_1 \rightarrow$ vecinos no visitados: $v_2, v_3 \rightarrow$ Cola: $[v_2, v_3]$

Procesamos: $v_2 \rightarrow$ vecinos no visitados: $v_4 \rightarrow$ Cola: $[v_3, v_4]$

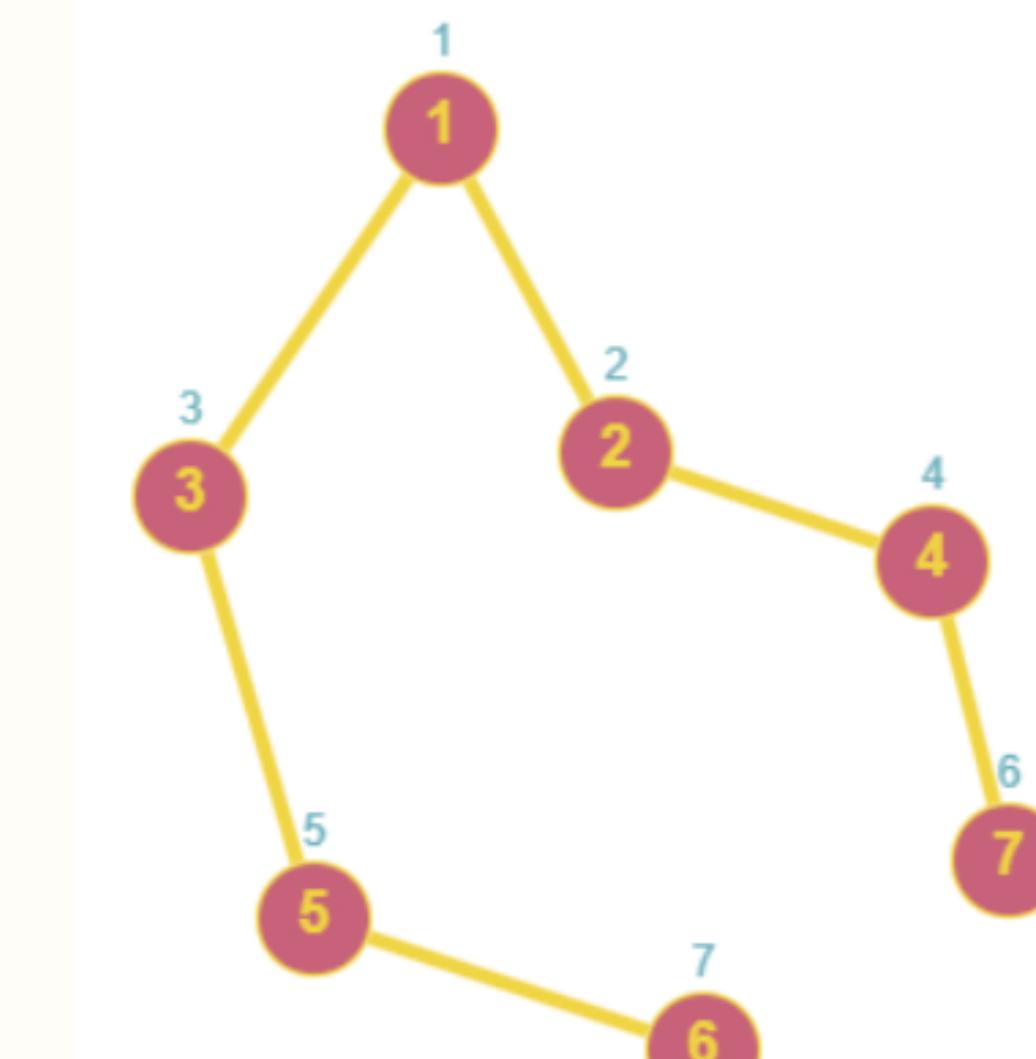
Procesamos: $v_3 \rightarrow$ vecinos no visitados: $v_5 \rightarrow$ Cola: $[v_4, v_5]$

Procesamos: $v_4 \rightarrow$ vecinos no visitados: $v_7 \rightarrow$ Cola: $[v_5, v_7]$

Procesamos: $v_5 \rightarrow$ vecinos no visitados: $v_6 \rightarrow$ Cola: $[v_7, v_6]$

Procesamos: $v_7 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Cola: $[v_6]$

Procesamos: $v_6 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Cola vacía.



Ejercicio

Realizar un recorrido en anchura iniciando desde el nodo v1 usando la siguiente matriz de adyacencia:

G(V,A)

V:{ 1, 2, 3, 4, 5, 6, 7}

**A:{(1,2), (1,3), (2,1), (3,1), (3,5), (2,4),
(4,2), (5,3), (5,6), (6,5),(6,7), (7,6),
(4,6), (6,4)}**

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	1	0	0	0	0
v2	1	0	0	1	0	0	0
v3	1	0	0	0	1	0	0
v4	0	1	0	0	0	1	0
v5	0	0	1	0	0	1	0
v6	0	0	0	1	1	0	1
v7	0	0	0	0	0	1	0

Solución

Recorrido final:

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_7$

Inicio en $v_1 \rightarrow$ Cola: [1]

Procesamos: $v_1 \rightarrow$ vecinos no visitados: $v_2, v_3 \rightarrow$ Cola: [2,3]

Procesamos: $v_2 \rightarrow$ vecinos no visitados: $v_4 \rightarrow$ Cola: [3,4]

Procesamos: $v_3 \rightarrow$ vecinos no visitados: $v_5 \rightarrow$ Cola: [4,5]

Procesamos: $v_4 \rightarrow$ vecinos no visitados: $v_6 \rightarrow$ Cola: [5,6]

Procesamos: $v_5 \rightarrow$ vecinos no visitados: 0 \rightarrow Cola: [6]

Procesamos: $v_6 \rightarrow$ vecinos no visitados: $v_7 \rightarrow$ Cola: [7]

Procesamos: $v_7 \rightarrow$ vecinos no visitados: 0 \rightarrow Cola vacía.

Árbol de expansión:

M	v_1	v_2	v_3	v_4	v_5	v_6	v_7
v_1	0	1	1	0	0	0	0
v_2	1	0	0	1	0	0	0
v_3	1	0	0	0	1	0	0
v_4	0	1	0	0	0	1	0
v_5	0	0	1	0	0	0	0
v_6	0	0	0	1	0	0	1
v_7	0	0	0	0	0	1	0

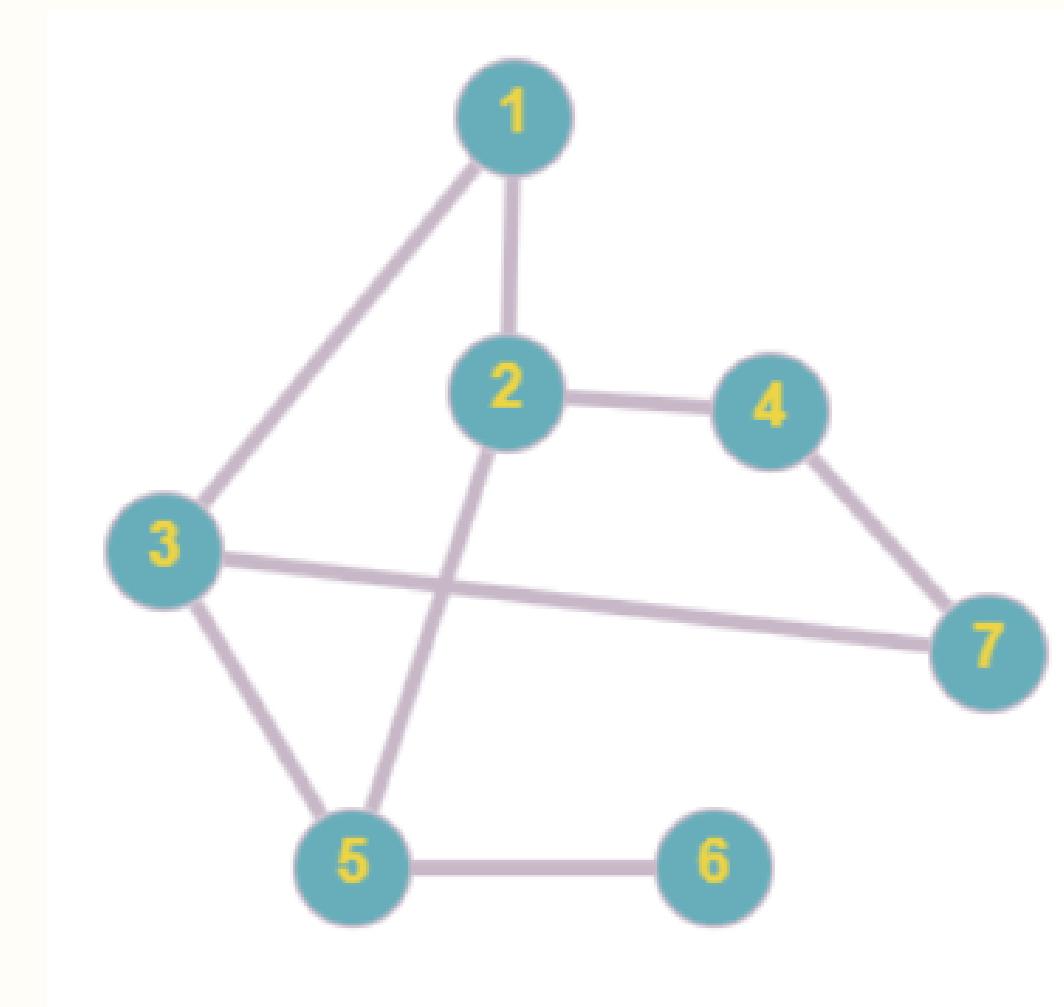
Ejercicio

Realizar un recorrido en profundidad iniciando desde el nodo v1 del siguiente grafo:

G(V,A)

V:{ 1, 2, 3, 4, 5, 6, 7}

A:{(1,2), (1,3), (2,1), (3,1), (3,5), (2,4),
(4,2), (5,3), (5,6), (6,5),(6,7), (7,6),
(4,6), (6,4), (3,7), (7,3), (2,5), (5,2)}



Solución

Recorrido final:

$v_1 \rightarrow v_3 \rightarrow v_7 \rightarrow v_4 \rightarrow v_5 \rightarrow v_6 \rightarrow v_2$

Inicio en $v_1 \rightarrow$ Pila: $[v_1]$

Procesamos: $v_1 \rightarrow$ vecinos no visitados: $v_2, v_3 \rightarrow$ Pila: $[v_3, v_2]$

Procesamos: $v_3 \rightarrow$ vecinos no visitados: $v_5, v_7 \rightarrow$ Pila: $[v_7, v_5, v_2]$

Procesamos: $v_7 \rightarrow$ vecinos no visitados: $v_4 \rightarrow$ Pila: $[v_4, v_5, v_2]$

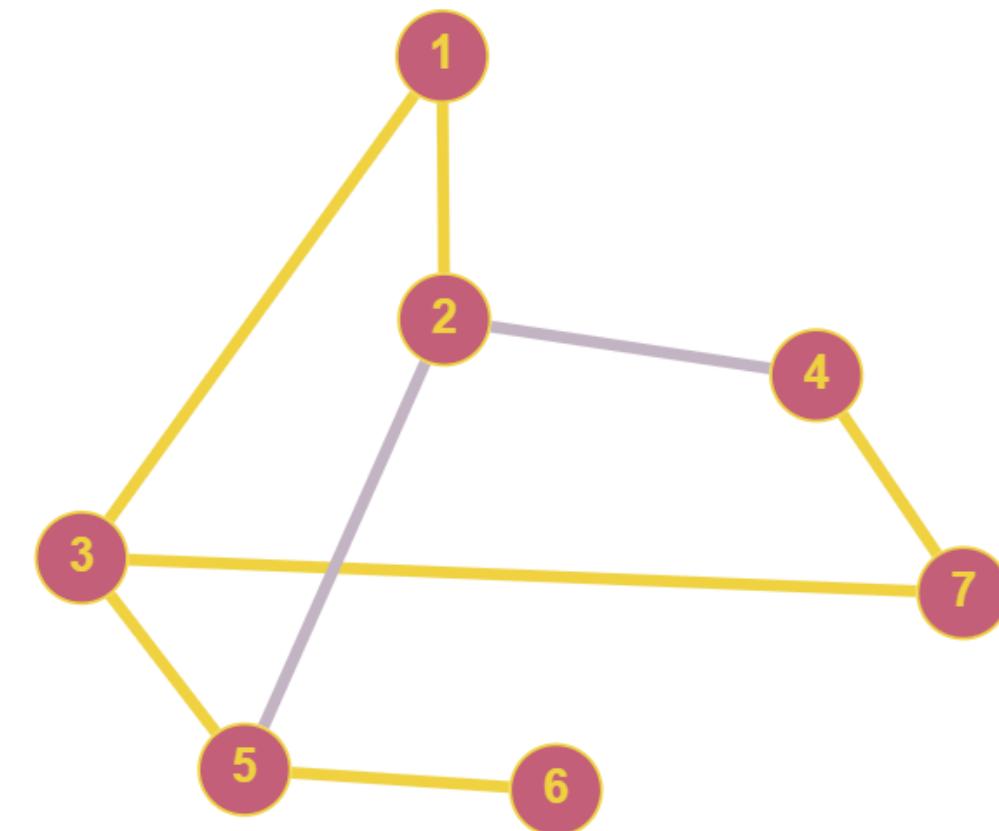
Procesamos: $v_4 \rightarrow$ vecinos no visitados: $v_0 \rightarrow$ Pila: $[v_5, v_2]$

Procesamos: $v_5 \rightarrow$ vecinos no visitados: $v_6 \rightarrow$ Pila: $[v_6, v_2]$

Procesamos: $v_6 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Pila: $[v_2]$

Procesamos: $v_2 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Pila vacía.

Árbol de expansión:



Ejercicio

Realizar un recorrido en anchura iniciando desde el nodo v1 usando la siguiente lista de adyacencia:

Lista de adyacencia:

v1 → (v2) → (v3)

v2 → (v1) → (v4) → (v6)

v3 → (v1) → (v6)

v4 → (v2) → (v5)

v5 → (v4)

v6 → (v2) → (v3)

G(V,A)

V:{ 1, 2, 3, 4, 5, 6 }

A:{(1,2), (1,3), (2,1), (2,4), (2,6), (3,1),
(3,6), (4,2), (4,5), (5,4),(6,2), (6,3) }

Solución

Recorrido final en BFS:

$v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_6 \rightarrow v_5$

Inicio en $v_1 \rightarrow$ Pila: $[v_1]$

Procesamos: $v_1 \rightarrow$ vecinos no visitados: $v_2, v_3 \rightarrow$ Pila: $[v_2, v_3]$

Procesamos: $v_2 \rightarrow$ vecinos no visitados: $v_4, v_6 \rightarrow$ Pila: $[v_3, v_4, v_6]$

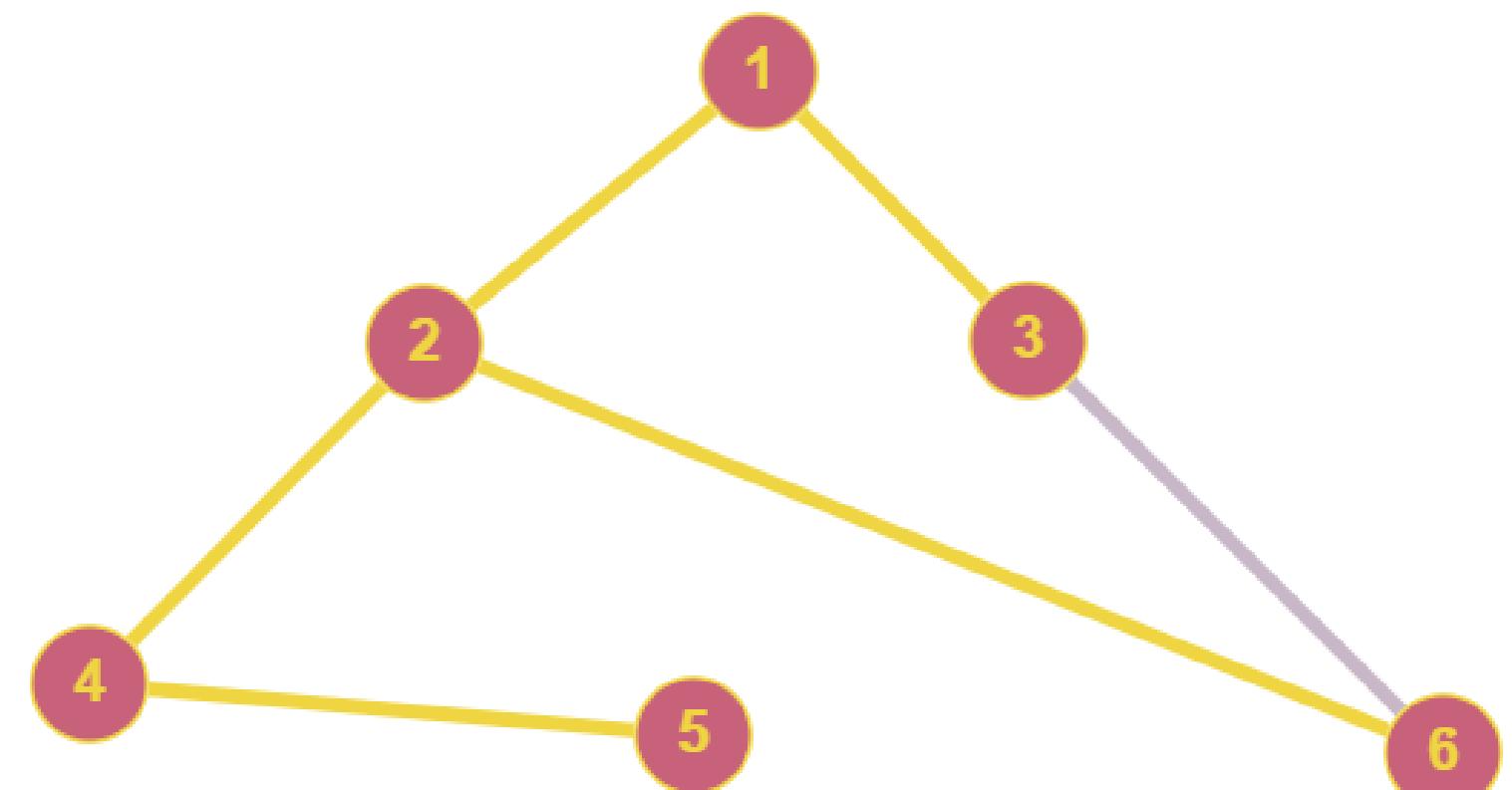
Procesamos: $v_3 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Pila: $[v_4, v_6]$

Procesamos: $v_4 \rightarrow$ vecinos no visitados: $v_5 \rightarrow$ Pila: $[v_6, v_5]$

Procesamos: $v_6 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Pila: $[v_5]$

Procesamos: $v_5 \rightarrow$ vecinos no visitados: $0 \rightarrow$ Pila: $[]$

Árbol de expansión:





04

Árbol de expansión mínima en grafos ponderados

Árbol de expansión mínima

Es un subgrafo que no tiene ciclos, conecta todos los vértices del grafo original y la suma de sus pesos es la menor posible entre todas sus combinaciones.

El algoritmo de Prim

- Se guarda el nodo actual en el árbol de expansión.
- Agrega todas las aristas de ese nodo a la lista que no estén en ella y que no sea la ultima agregada al árbol, en caso de que la arista este en la lista, se eliminan de esta.
- Se selecciona la arista con menor peso dentro de la lista.
- Se guarda esta arista en el árbol y se elimina de la lista.
- Se selecciona el nodo al que lleva la arista como nodo actual.
- Se repite hasta que no haya elementos en la lista o el árbol contenga a todos los nodos

Notas:

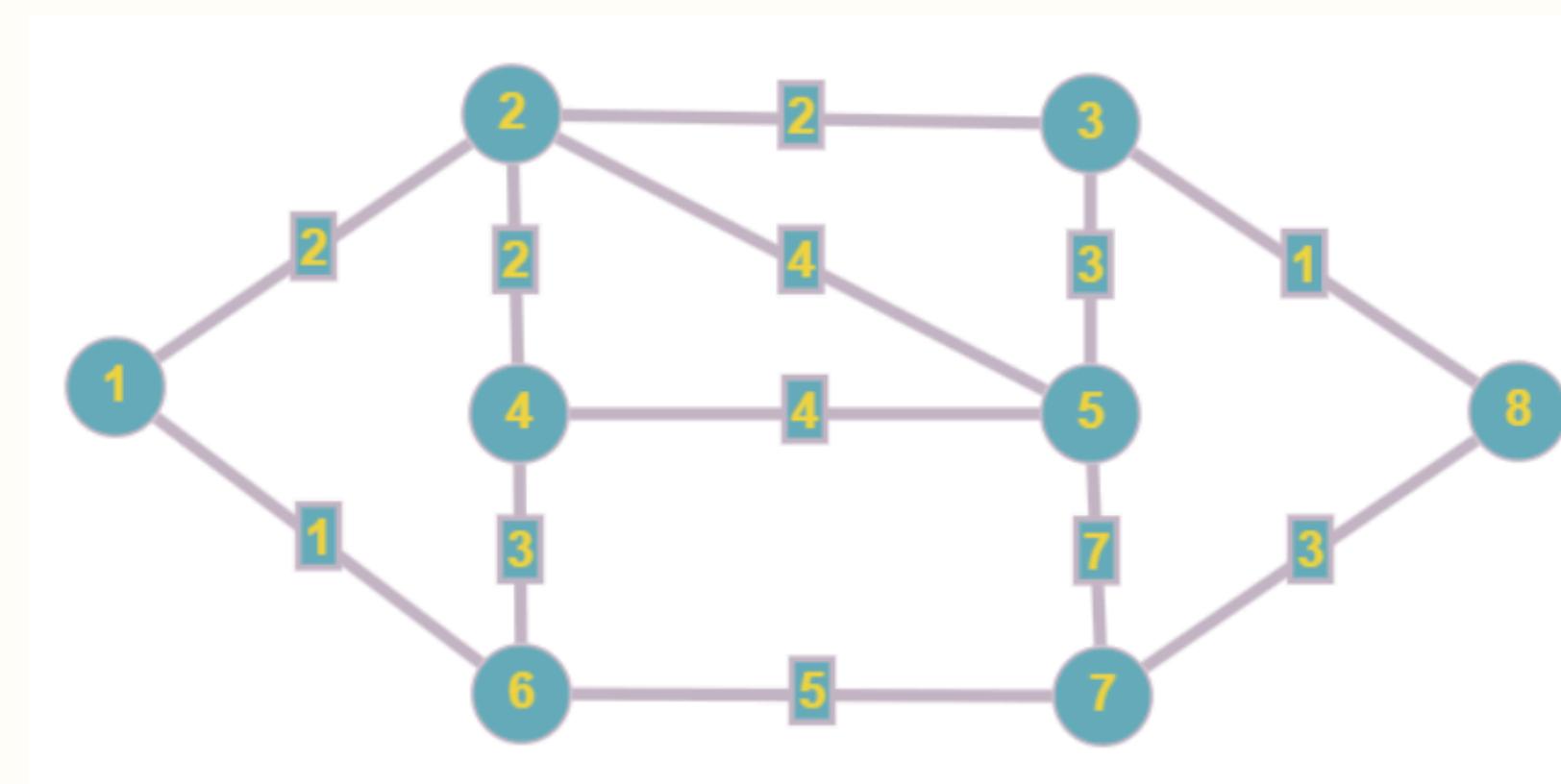
- El grafo debe de ser convexo
- La lista empieza vacía
- El árbol de expansión puede cambiar dependiendo del nodo inicial
- La suma de los pesos será la misma para todos los arboles de expansión posibles

Utilizado para
grafos densos

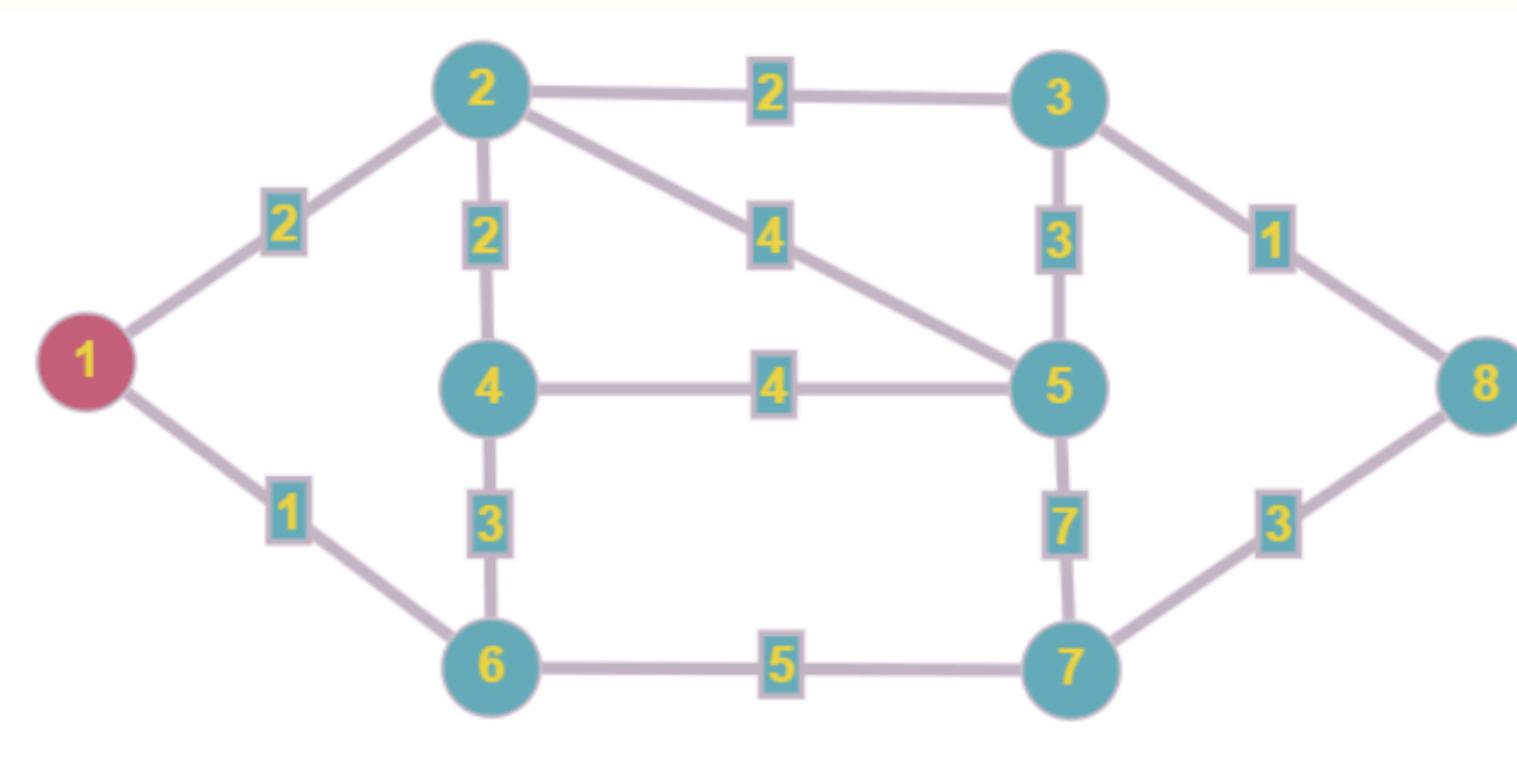
Ejemplo



Realizar el algoritmo de Prim empezando del vértice v1.



Algoritmo de Prim



Se guarda el nodo v_1 en el árbol de expansión.

Se guardan sus aristas a la lista de manera ordenada
[1 (v_1, v_6)
2 (v_1, v_2)]

Algoritmo de Prim

Se toma de la lista el primer elemento [el menor]:

1 (v1, v6)

Se elimina de la lista y se agrega al árbol

Se selecciona v6 como nodo actual y se agrega al árbol.

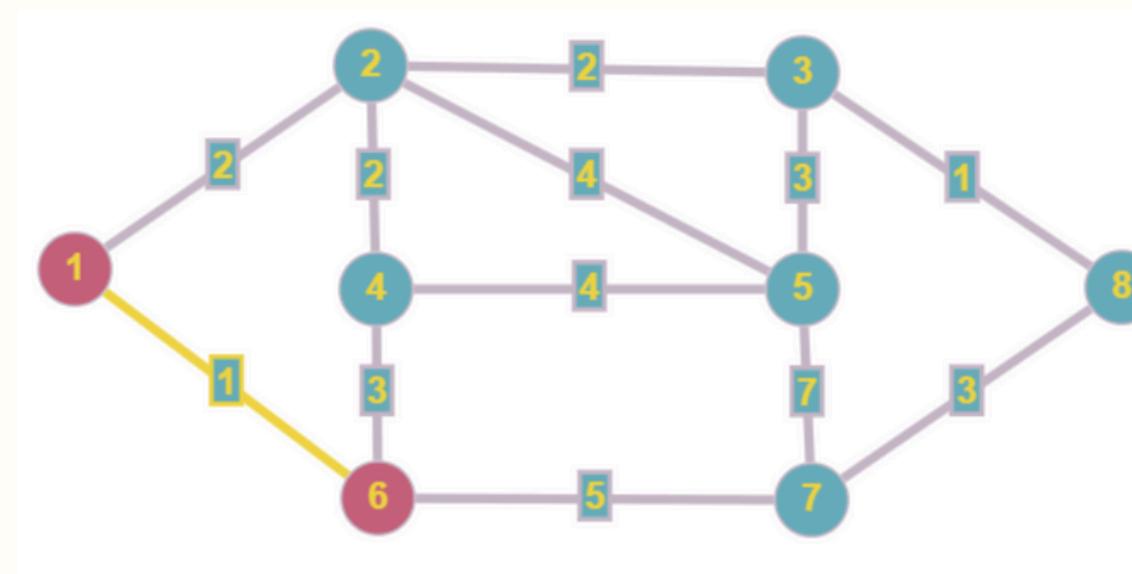
Se guardan sus aristas a la lista de manera ordenada

[2 (v1, v2)

3 (v4, v6)

5 (v6, v7)]

La arista 1 (v1, v6) fue el ultimo agregado entonces se ignora.



Algoritmo de Prim

Se toma de la lista el primer elemento [el menor]:

2 (v1, v2)

Se elimina de la lista y se agrega al árbol

Se selecciona v2 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[2 (v2, v4)

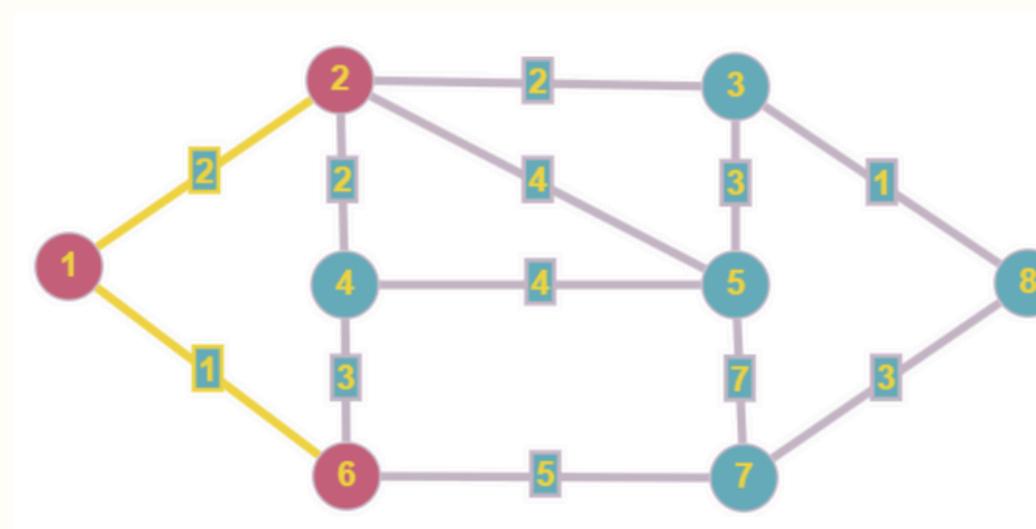
2 (v2, v3)

3 (v4, v6)

4 (v2, v5)

5 (v6, v7)]

La arista 2 (v1, v2) fue el ultimo agregado entonces se ignora.



Algoritmo de Prim

Se toma de la lista el primer elemento [el menor]:
2 (v2, v4)

Se elimina de la lista y se agrega al árbol

Se selecciona v4 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[2 (v2, v3)

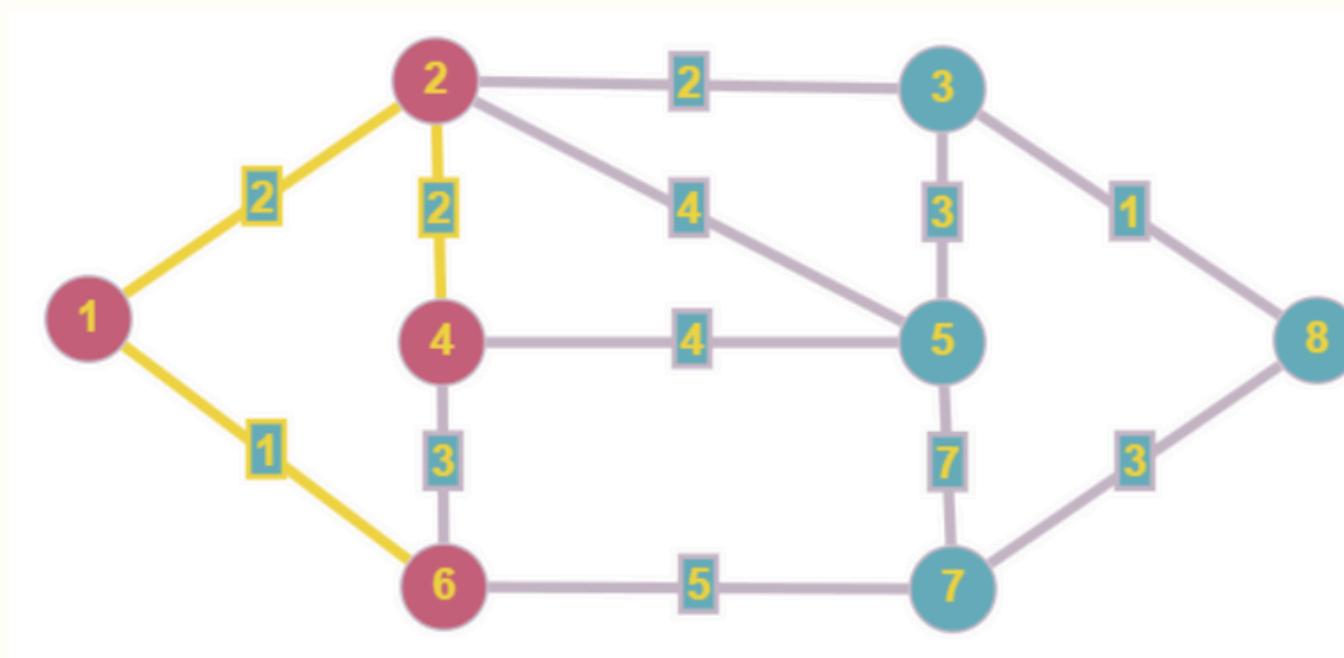
4 (v2, v5)

4 (v4, v5)

5 (v6, v7)]

La arista 2 (v2, v4) fue el ultimo agregado entonces se ignora.

La arista 3 (v4, v6), esta en la lista y se elimina



Algoritmo de Prim

Se toma de la lista el primer elemento [el menor]:
2 (v2, v3)

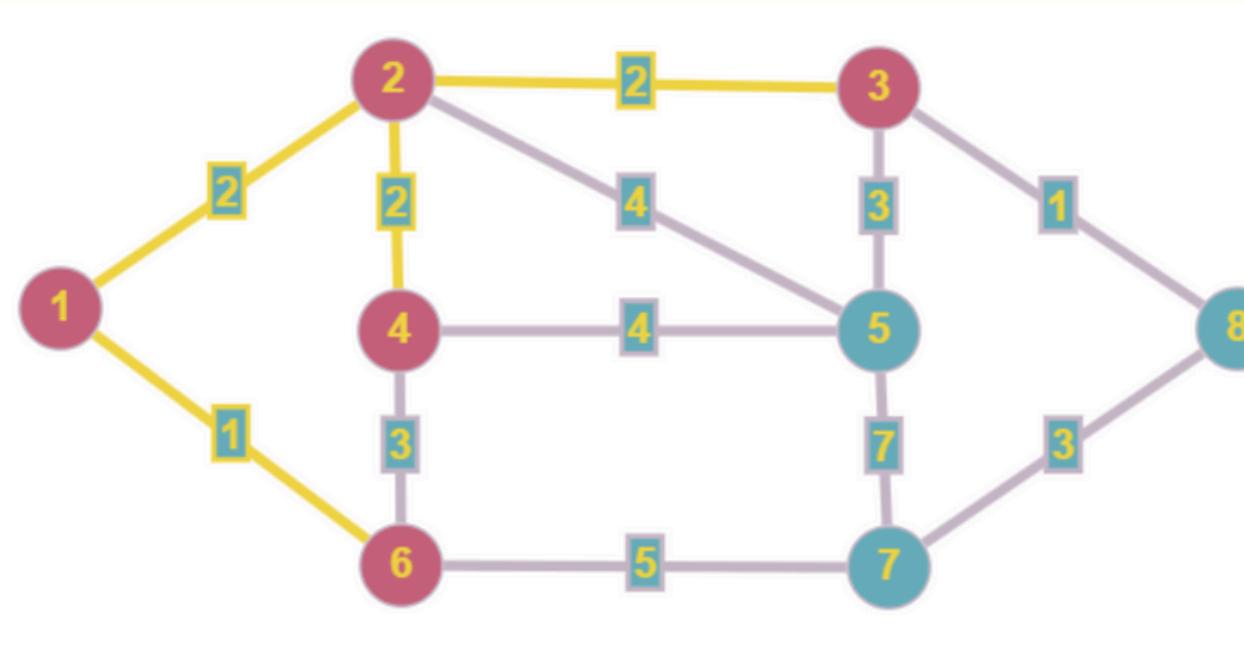
Se elimina de la lista y se agrega al árbol

Se selecciona v3 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[1 (v3, v8)
3 (v3, v5)
4 (v2, v5)
4 (v4, v5)
5 (v6, v7)]

La arista 2 (v2, v3) fue el ultimo agregado entonces se ignora.



Algoritmo de Prim

Se toma de la lista el primer elemento [el menor]:

1 (v3, v8)

Se elimina de la lista y se agrega al árbol

Se selecciona v8 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[3 (v3, v5)

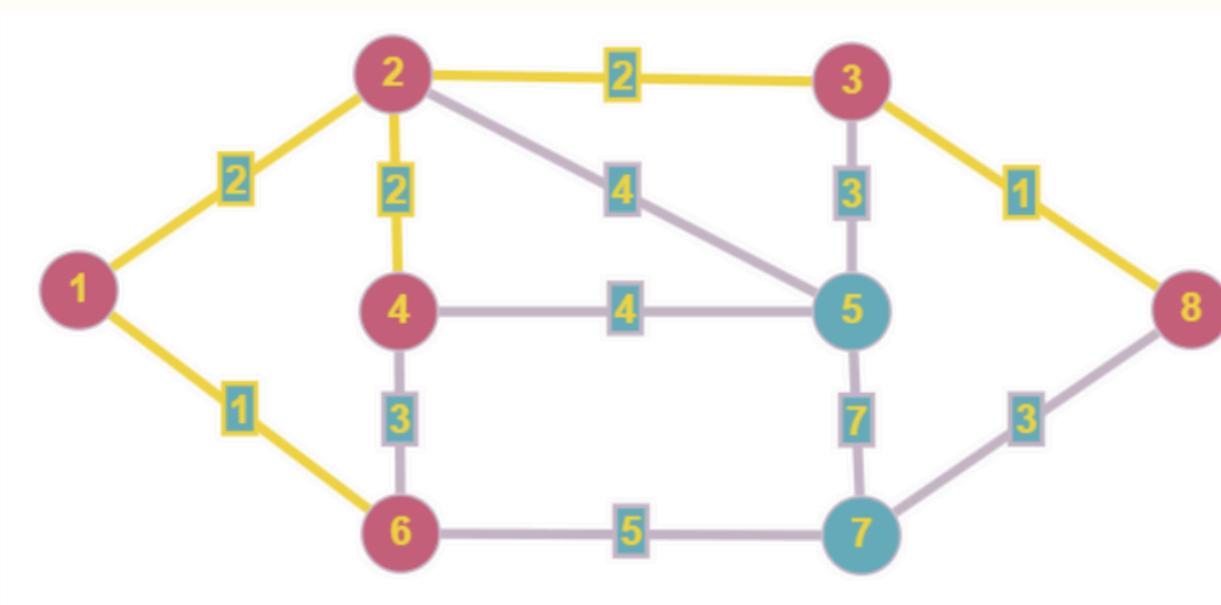
3 (v7, v8)

4 (v2, v5)

4 (v4, v5)

5 (v6, v7)]

La arista 2 (v3, v8) fue el ultimo agregado entonces se ignora.



Algoritmo de Prim

Se toma de la lista el primer elemento [el menor]:
3 (v3, v5)

Se elimina de la lista y se agrega al árbol

Se selecciona v5 como nodo actual y se agrega al árbol.

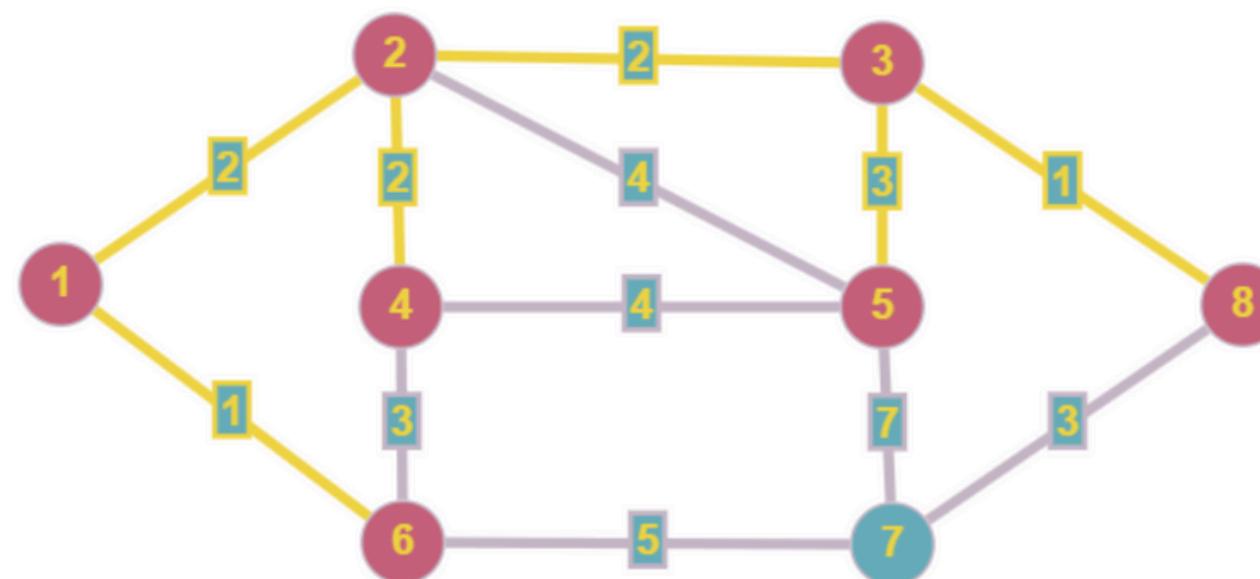
Se guardan sus aristas a la lista de manera ordenada

[3 (v7, v8)
5 (v6, v7)
7 (v5, v7)]

La arista 4 (v2, v5) esta en la lista y se elimina.

La arista 3 (v3, v5) fue el ultimo agregado entonces se ignora.

La arista 4 (v4, v5) esta en la lista y se elimina.



Algoritmo de Prim

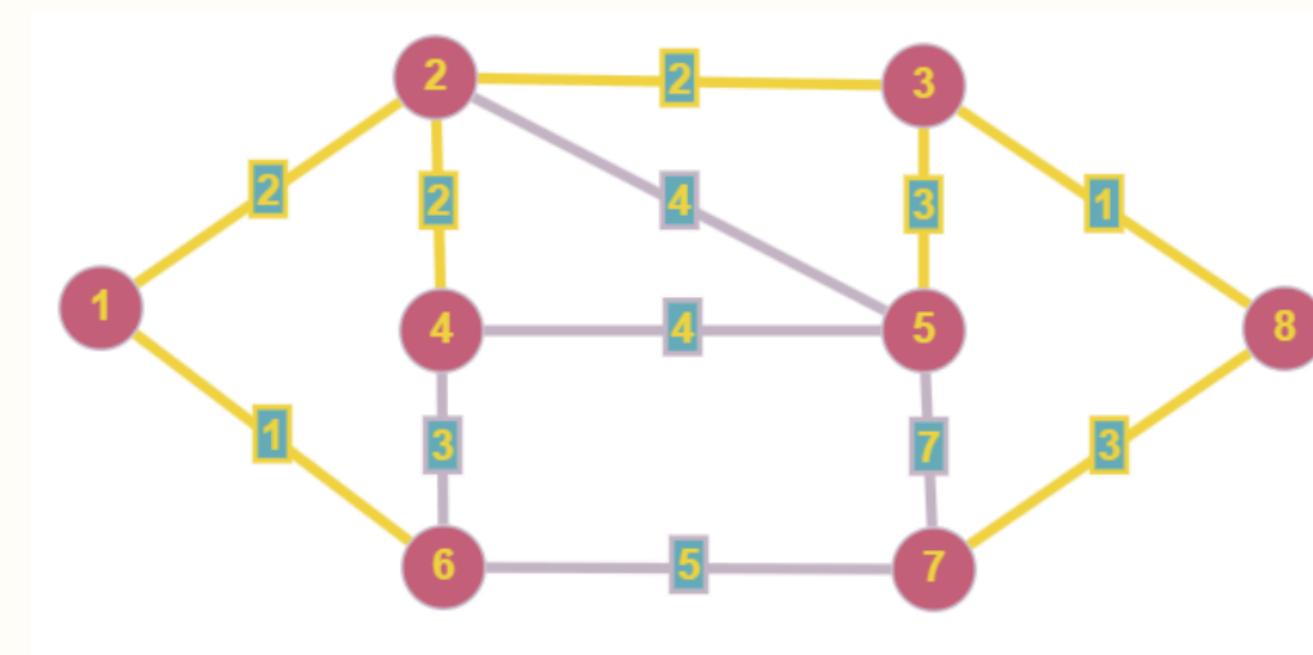
Se toma de la lista el primer elemento [el menor]:
3 (v7, v8)

Se elimina de la lista y se agrega al árbol

Se selecciona v7 como nodo actual y se agrega al árbol.

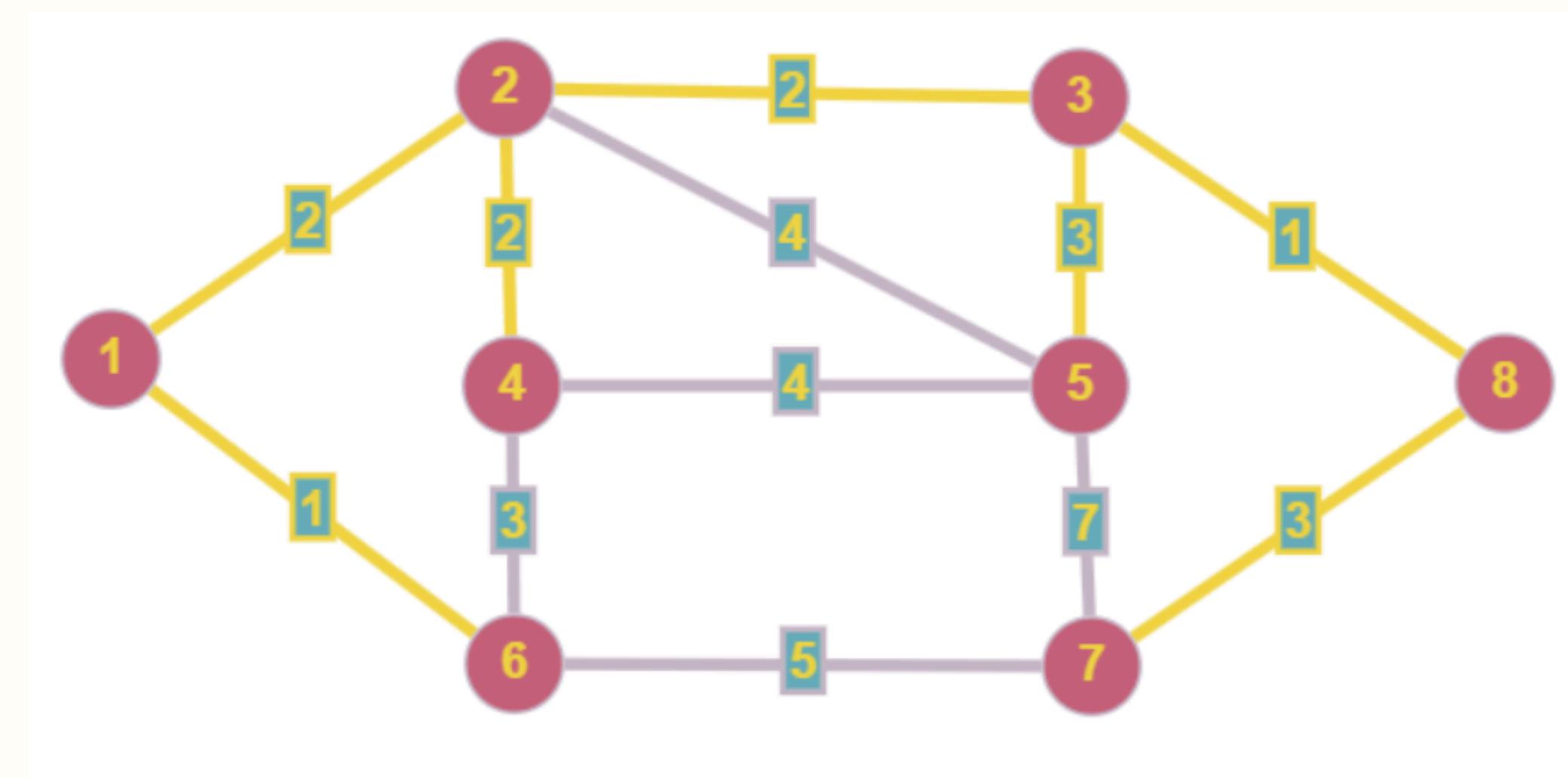
Se guardan sus aristas a la lista de manera ordenada
[]

La arista 5 (v6, v7) esta en la lista y se elimina.
La arista 7 (v5, v7) esta en la lista y se elimina.



Algoritmo de Prim

Como la lista quedó vacía, se termina el algoritmo



Ejemplo

Realizar el algoritmo de Prim empezando del vértice v6.

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	3	∞	1	∞	1
v2	1	0	∞	7	2	∞	∞
v3	3	∞	0	1	3	∞	5
v4	∞	7	1	0	∞	3	5
v5	1	2	3	∞	0	∞	4
v6	∞	∞	∞	3	∞	0	8
v7	1	∞	5	5	4	8	0

Ejemplo

M	V1	V2	V3	V4	V5	V6	V7
V1	0	1	3	∞	1	∞	1
V2	1	0	∞	7	2	∞	∞
V3	3	∞	0	1	3	∞	5
V4	∞	7	1	0	∞	3	5
V5	1	2	3	∞	0	∞	4
V6	∞	∞	∞	3	∞	0	8
V7	1	∞	5	5	4	8	0

M	V6
V6	0

Se guarda el nodo v6 en el árbol de expansión.

**Se guardan sus aristas a la lista de manera ordenada
[3 (v4, v6)
8 (v6, v7)]**

Ejemplo

M	V1	V2	V3	V4	V5	V6	V7
V1	0	1	3	∞	1	∞	1
V2	1	0	∞	7	2	∞	∞
V3	3	∞	0	1	3	∞	5
V4	∞	7	1	0	∞	3	5
V5	1	2	3	∞	0	∞	4
V6	∞	∞	∞	3	∞	0	8
V7	1	∞	5	5	4	8	0

M	V6	V4
V6	0	3
v4	3	0

Se toma de la lista el primer elemento [el menor]:

3 (v4, v6)

Se elimina de la lista y se agrega al árbol

Se selecciona v4 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[1 (v3, v4)
5 (v4, v7)
7 (v2, v4)
8 (v6, v7)]

La arista 3 (v4, v6) fue el ultimo agregado entonces se ignora.

Ejemplo

M	V1	V2	V3	V4	V5	V6	V7
V1	0	1	3	∞	1	∞	1
V2	1	0	∞	7	2	∞	∞
V3	3	∞	0	1	3	∞	5
V4	∞	7	1	0	∞	3	5
V5	1	2	3	∞	0	∞	4
V6	∞	∞	∞	3	∞	0	8
V7	1	∞	5	5	4	8	0

M	V6	V4	V3
V6	0	3	∞
v4	3	0	1
V3	∞	1	0

Se toma de la lista el primer elemento [el menor]:

1 (v3, v4)

Se elimina de la lista y se agrega al árbol

Se selecciona v3 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[3 (v1, v3)
3 (v3, v5)
5 (v3, v7)
5 (v4, v7)
7 (v2, v4)
8 (v6, v7)]

La arista 1 (v3, v4) fue el ultimo agregado entonces se ignora.

Ejemplo

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	3	∞	1	∞	1
v2	1	0	∞	7	2	∞	∞
v3	3	∞	0	1	3	∞	5
v4	∞	7	1	0	∞	3	5
v5	1	2	3	∞	0	∞	4
v6	∞	∞	∞	3	∞	0	8
v7	1	∞	5	5	4	8	0

M	v6	v4	v3	v1
v6	0	3	∞	∞
v4	3	0	1	∞
v3	∞	1	0	3
v1	∞	∞	3	0

Se toma de la lista el primer elemento [el menor]:

3 (v1, v3)

Se elimina de la lista y se agrega al árbol

Se selecciona v1 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[1 (v1, v2)
 1 (v1, v5)
 1 (v1, v7)
 3 (v3, v5)
 5 (v3, v7)
 5 (v4, v7)
 7 (v2, v4)
 8 (v6, v7)]

La arista 3 (v1, v3) fue el ultimo agregado entonces se ignora.

Ejemplo

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	3	∞	1	∞	1
v2	1	0	∞	7	2	∞	∞
v3	3	∞	0	1	3	∞	5
v4	∞	7	1	0	∞	3	5
v5	1	2	3	∞	0	∞	4
v6	∞	∞	∞	3	∞	0	8
v7	1	∞	5	5	4	8	0

La arista 1 (v1, v2) fue el ultimo agregado entonces se ignora.

M	v6	v4	v3	v1	v2
v6	0	3	∞	∞	∞
v4	3	0	1	∞	∞
v3	∞	1	0	3	∞
v1	∞	∞	3	0	1
v2	∞	∞	∞	1	0

Se toma de la lista el primer elemento [el menor]:

1 (v1, v2)

Se elimina de la lista y se agrega al árbol

Se selecciona v2 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[1 (v1, v5)

1 (v1, v7)

2 (v2, v5)

3 (v3, v5)

5 (v3, v7)

5 (v4, v7)

7 (v2, v4) / 7 (v2, v4)

8 (v6, v7)]

Se eliminan las aristas repetidas

Ejemplo

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	3	∞	1	∞	1
v2	1	0	∞	7	2	∞	∞
v3	3	∞	0	1	3	∞	5
v4	∞	7	1	0	∞	3	5
v5	1	2	3	∞	0	∞	4
v6	∞	∞	∞	3	∞	0	8
v7	1	∞	5	5	4	8	0

La arista 1 (v1, v5) fue el ultimo agregado entonces se ignora.

M	v6	v4	v3	v1	v2	v5
v6	0	3	∞	∞	∞	∞
v4	3	0	1	∞	∞	∞
v3	∞	1	0	3	∞	∞
v1	∞	∞	3	0	1	1
v2	∞	∞	∞	1	0	∞
v5	∞	∞	∞	1	∞	0

Se toma de la lista el primer elemento [el menor]:
1 (v1, v5)

Se elimina de la lista y se agrega al árbol

Se selecciona v5 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[

1 (v1, v7)

2 (v2, v5) // 2 (v2, v5)

3 (v3, v5) // 3 (v3, v5)

4 (v5, v7)

5 (v3, v7)

5 (v4, v7)

8 (v6, v7)]

Se eliminan las aristas repetidas

Ejemplo

M	v1	v2	v3	v4	v5	v6	v7
v1	0	1	3	∞	1	∞	1
v2	1	0	∞	7	2	∞	∞
v3	3	∞	0	1	3	∞	5
v4	∞	7	1	0	∞	3	5
v5	1	2	3	∞	0	∞	4
v6	∞	∞	∞	3	∞	0	8
v7	1	∞	5	5	4	8	0

La arista 1 (v1, v7) fue el ultimo agregado entonces se ignora.

M	v6	v4	v3	v1	v2	v5	v7
v6	0	3	∞	∞	∞	∞	∞
v4	3	0	1	∞	∞	∞	∞
v3	∞	1	0	3	∞	∞	∞
v1	∞	∞	3	0	1	1	1
v2	∞	∞	∞	1	0	∞	∞
v5	∞	∞	∞	1	∞	0	∞
v7	∞	∞	∞	1	∞	∞	0

Se toma de la lista el primer elemento [el menor]:

1 (v1, v7)

Se elimina de la lista y se agrega al árbol

Se selecciona v7 como nodo actual y se agrega al árbol.

Se guardan sus aristas a la lista de manera ordenada

[

1 (v1, v7) // 1 (v1, v7)

4 (v5, v7) // 4 (v5, v7)

5 (v3, v7) // 5 (v3, v7)

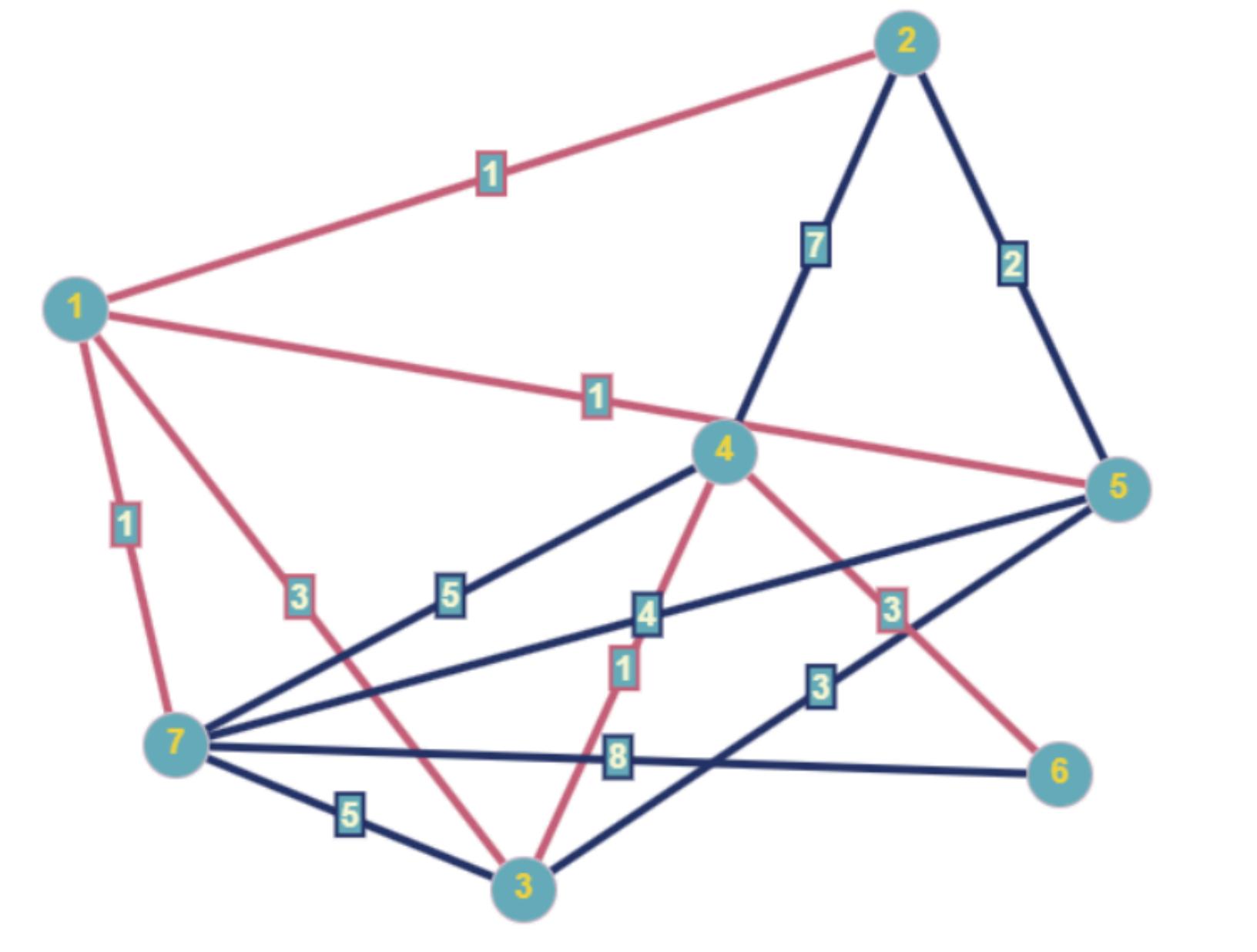
5 (v4, v7) // 5 (v4, v7)

8 (v6, v7) // 8 (v6, v7)]

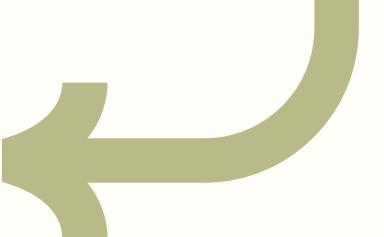
Se eliminan las aristas repetidas

Solucion

M	v6	v4	v3	v1	v2	v5	v7
v6	0	3	∞	∞	∞	∞	∞
v4	3	0	1	∞	∞	∞	∞
v3	∞	1	0	3	∞	∞	∞
v1	∞	∞	3	0	1	1	1
v2	∞	∞	∞	1	0	∞	∞
v5	∞	∞	∞	1	∞	0	∞
v7	∞	∞	∞	1	∞	∞	0



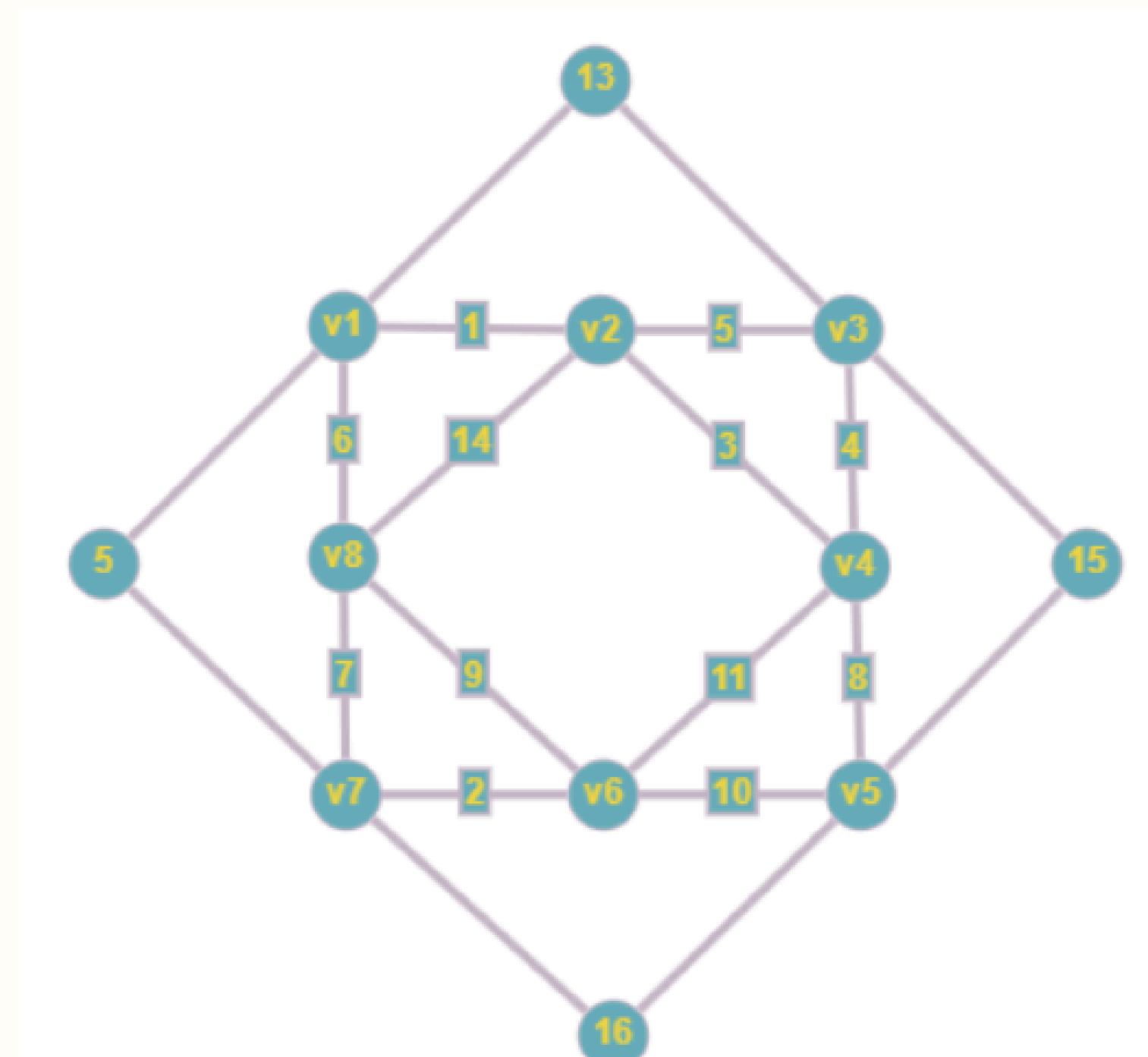
El algoritmo de kruskal



- Guarda en una lista todas las aristas de menor a mayor peso así como los nodos a los que viaja.
- Agarra el primer elemento de la lista
- Comprueba si alguno de los vértices no este en el conjunto de listas:
 - Si ninguno de los vértices esta en ninguna lista, se genera una nueva lista, se agregan ambos vértices y se agrega la arista al árbol
 - Si nada mas 1 solo vértice esta en una sola lista, se agrega el otro a la lista y se agrega la arista al árbol
 - Si un vértice esta en una lista y el otro en otra diferente, se juntan ambas listas y se agrega la arista al árbol
 - Si ambos vértices están en la misma lista, se desecha la arista
- Repetir hasta tener una sola lista con todos los vértices

Utilizado para
grafos dispersos

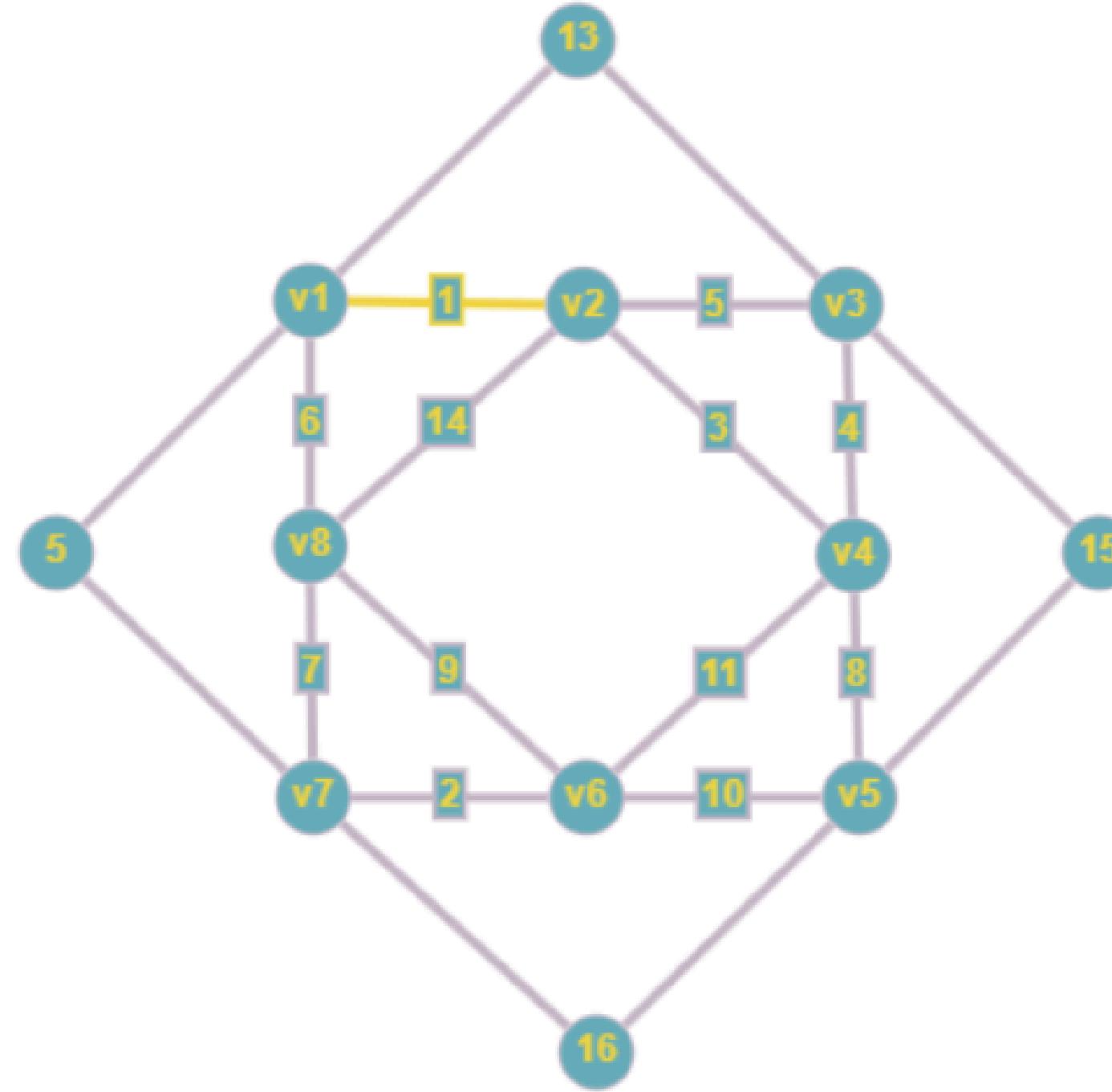
El algoritmo de kruskal



Lista de aristas:

- [1 (v1, v2)]
- 2 (v6, v7)
- 3 (v2, v4)
- 4 (v3, v4)
- 5 (v1, v7)
- 5 (v2, v3)
- 6 (v1, v8)
- 7 (v7, v8)
- 8 (v4, v5)
- 9 (v6, v8)
- 10 (v5, v6)
- 11 (v4, v6)
- 13 (v1, v3)
- 14 (v2, v8)
- 15 (v3, v5)
- 16 (v5, v7)]

El algoritmo de kruskal



Lista de aristas:

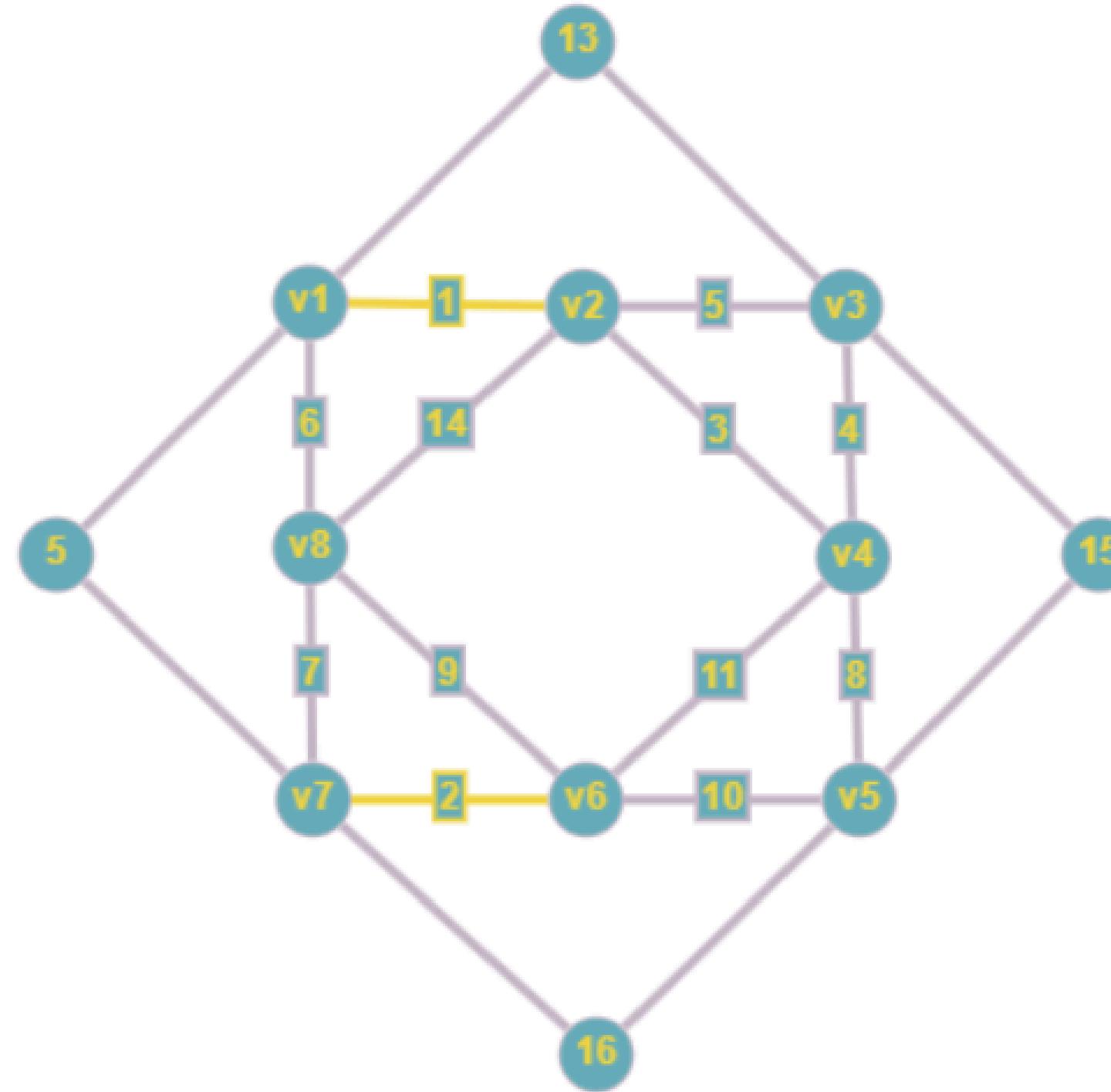
[2 (v6, v7)
3 (v2, v4)
4 (v3, v4)
5 (v1, v7)
5 (v2, v3)
6 (v1, v8)
7 (v7, v8)
8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

Se toma el elemento
1 (v1, v2)

Se crea la lista
[v1, v2]

Se agrega la arista

El algoritmo de kruskal



Lista de aristas:

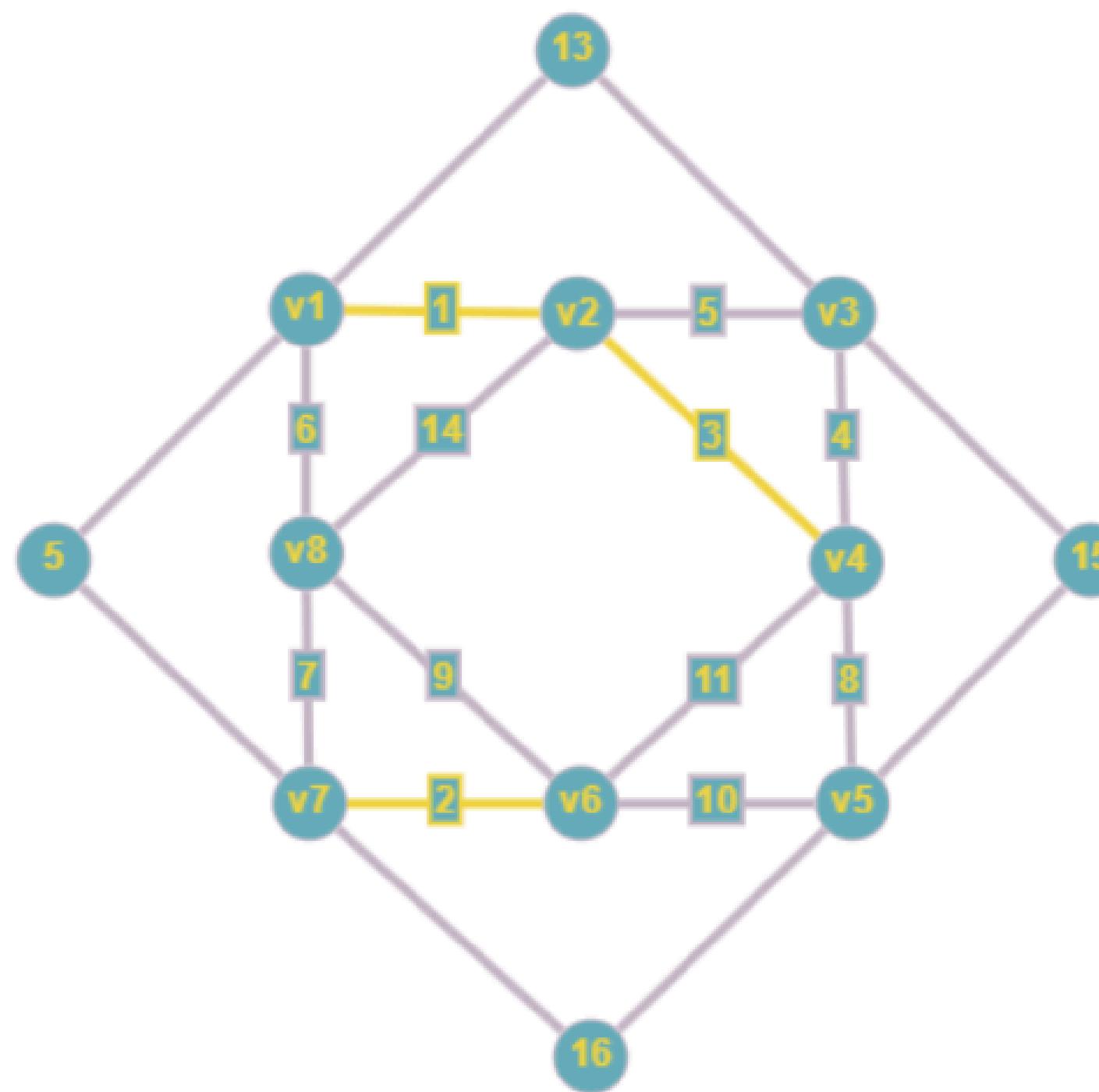
[3 (v2, v4)
4 (v3, v4)
5 (v1, v7)
5 (v2, v3)
6 (v1, v8)
7 (v7, v8)
8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

**Se toma el elemento
2 (v6, v7)**

**Se crea la lista
[v6, v7]
[v1, v2]**

Se agrega la arista

El algoritmo de kruskal



Lista de aristas:

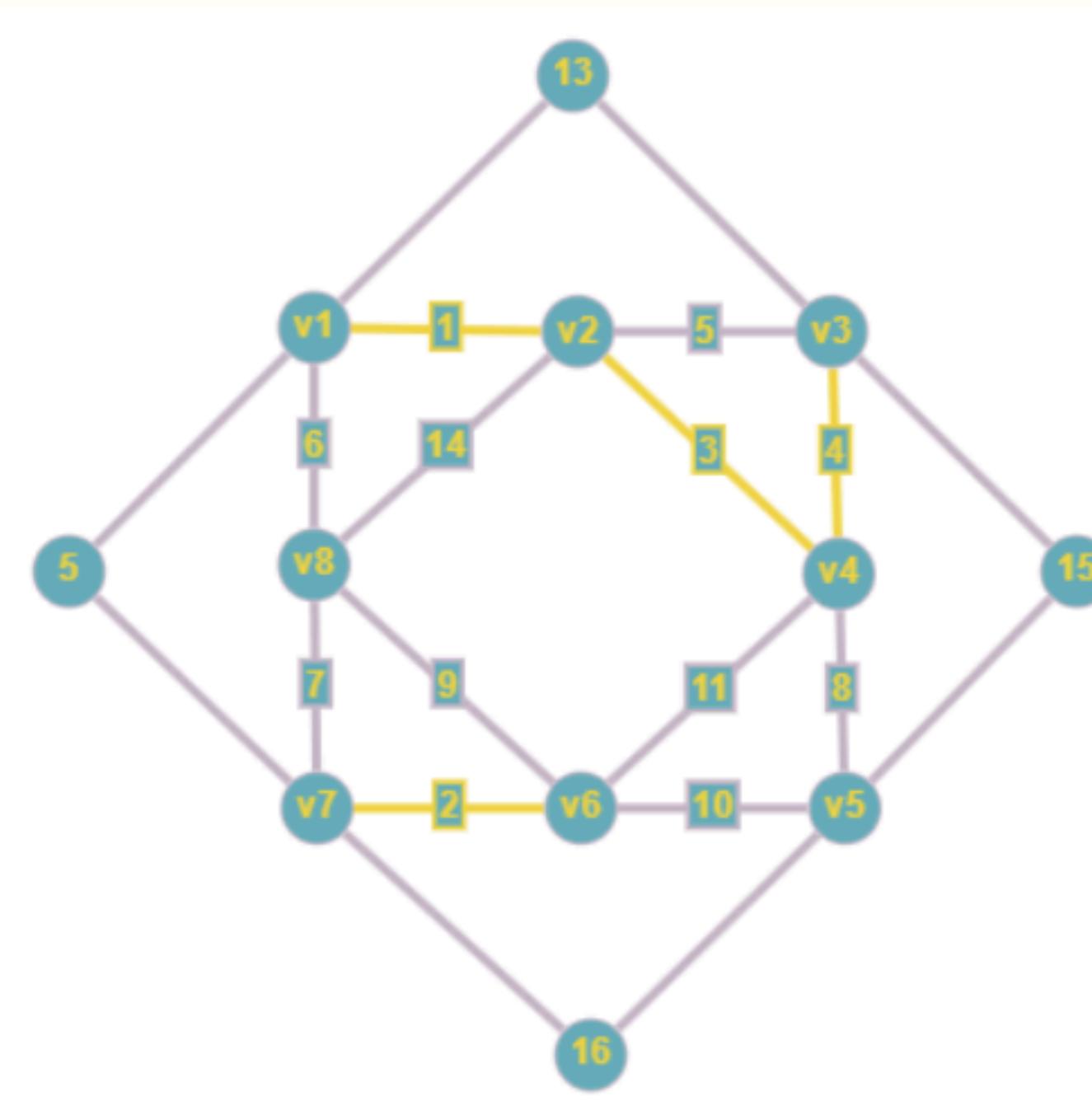
[4 (v3, v4)
5 (v1, v7)
5 (v2, v3)
6 (v1, v8)
7 (v7, v8)
8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

Se toma el elemento
3 (v2, v4)

Se agrega v4
[v6, v7]
[v1, v2, v4]

Se agrega la arista

El algoritmo de kruskal



Lista de aristas:

[5 (v1, v7)
5 (v2, v3)
6 (v1, v8)
7 (v7, v8)
8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

Se toma el elemento

4 (v3, v4)

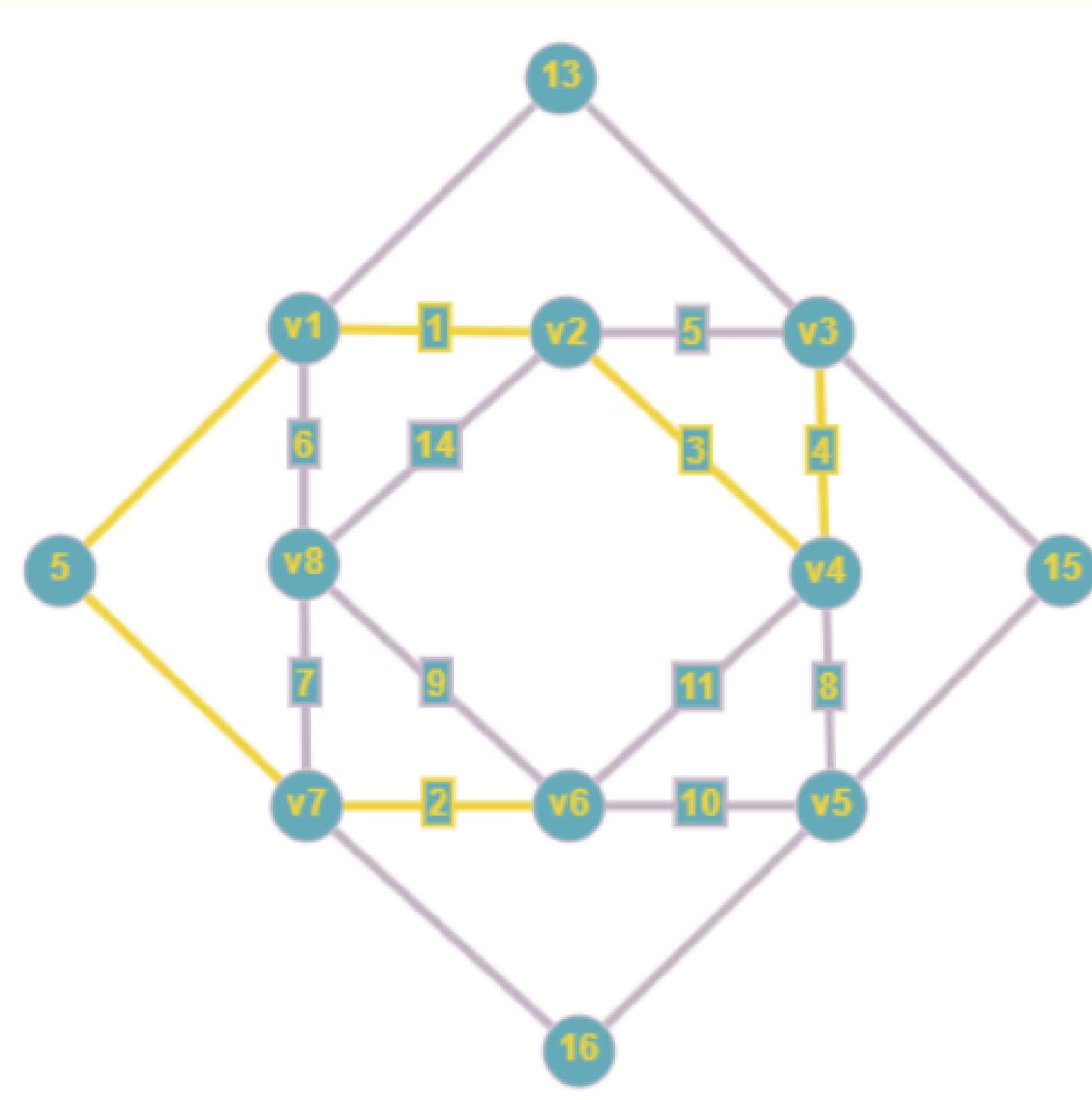
Se agrega v3

[v6, v7]

[v1, v2, v3, v4]

Se agrega la arista

El algoritmo de kruskal



Lista de aristas:

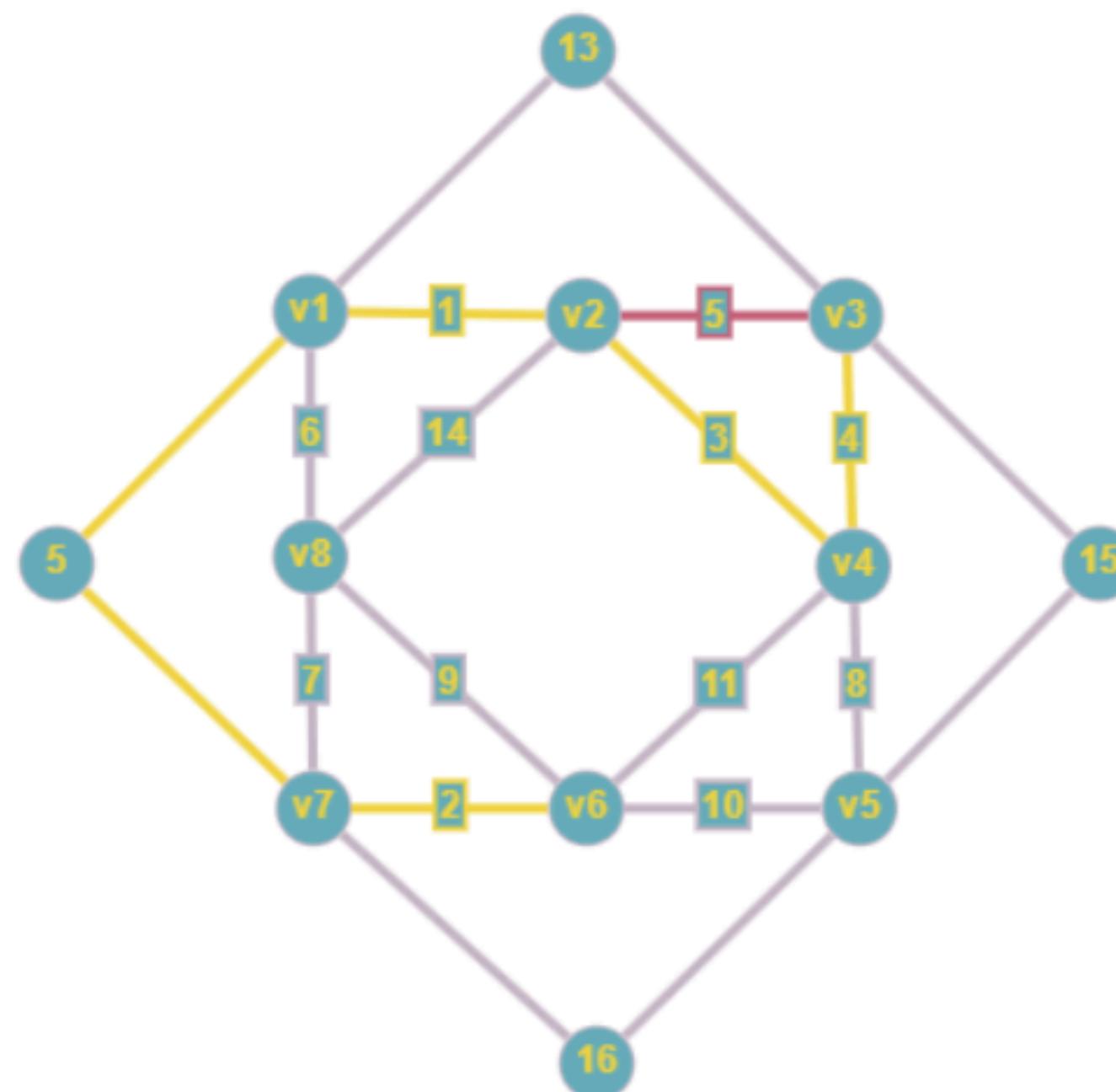
- [5 (v2, v3)]
- [6 (v1, v8)]
- [7 (v7, v8)]
- [8 (v4, v5)]
- [9 (v6, v8)]
- [10 (v5, v6)]
- [11 (v4, v6)]
- [13 (v1, v3)]
- [14 (v2, v8)]
- [15 (v3, v5)]
- [16 (v5, v7)]]

Se toma el elemento
5 (v1, v7)

Se juntan las listas
[v1, v2, v3, v4, v6, v7]

Se agrega la arista

El algoritmo de kruskal



Lista de aristas:

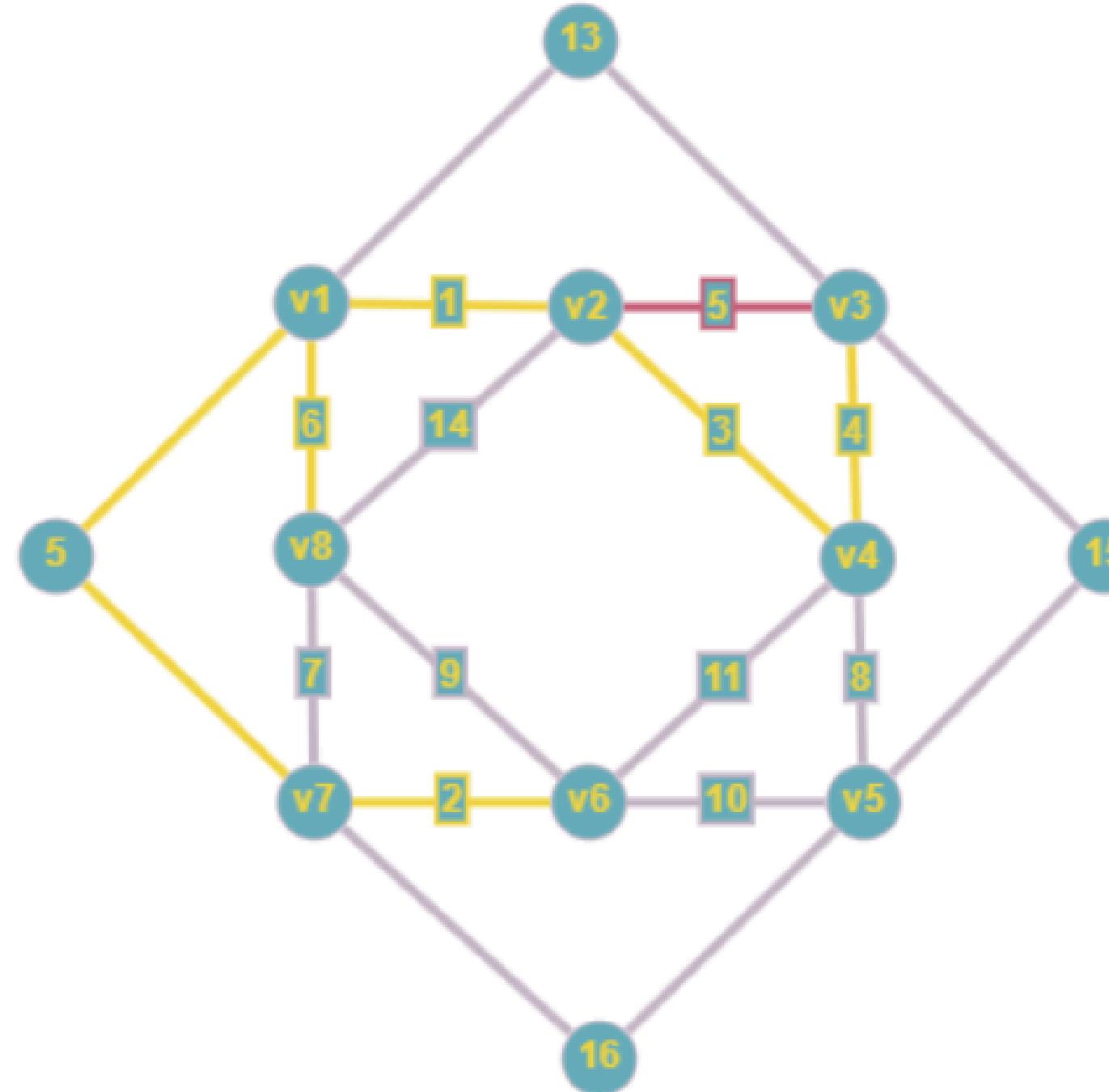
[6 (v1, v8)
7 (v7, v8)
8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

**Se toma el elemento
5 (v2, v3)**

**No se modifica nada
[v1, v2, v3, v4, v6, v7]**

**v2 y v3 ya están en la lista, no es
agrega la arista**

El algoritmo de kruskal



Lista de aristas:

[7 (v7, v8)
8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

Se toma el elemento

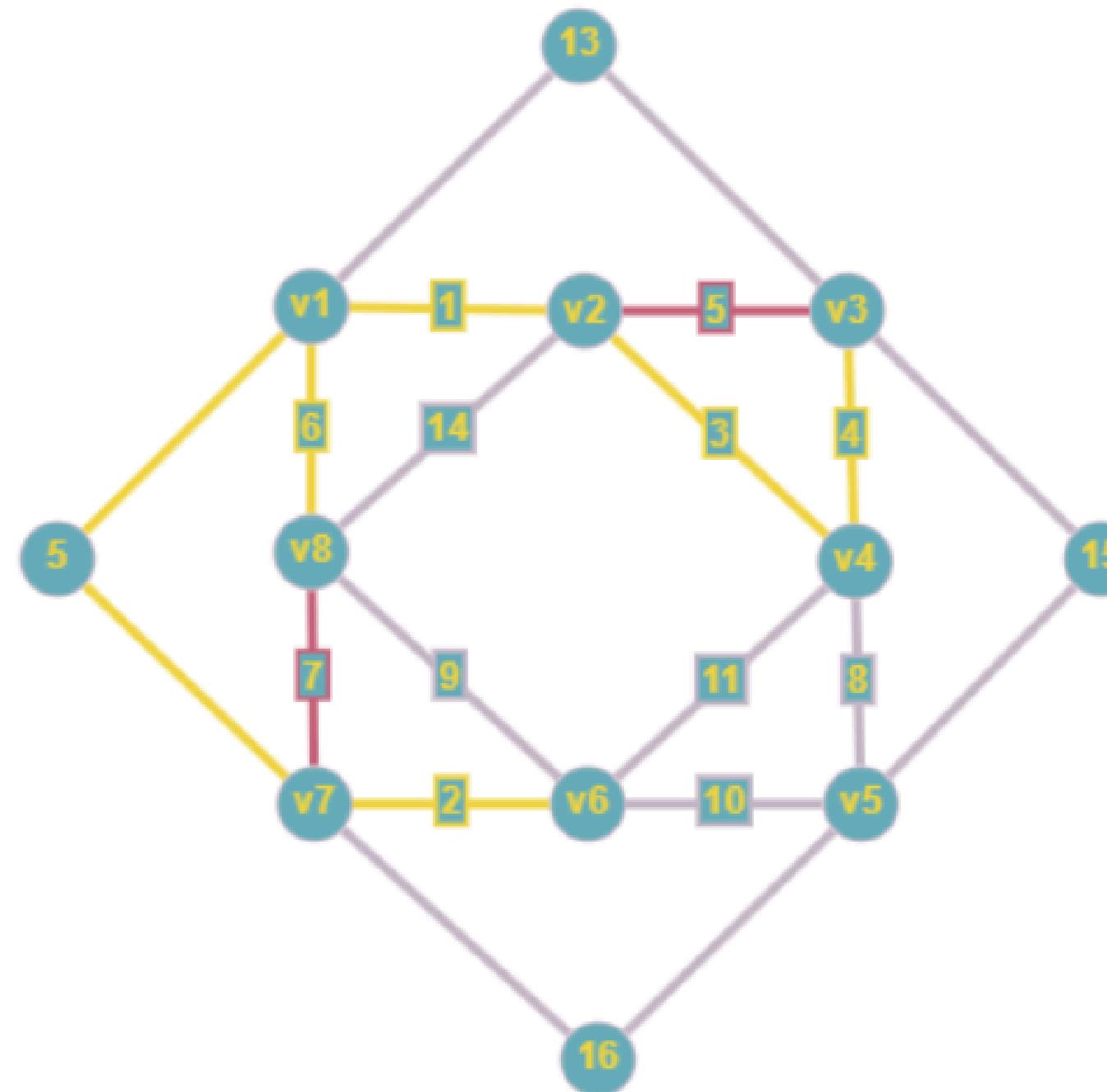
6 (v1, v8)

Se agrega v8

[v1, v2, v3, v4, v6, v7, v8]

Se agrega la arista

El algoritmo de kruskal



Lista de aristas:

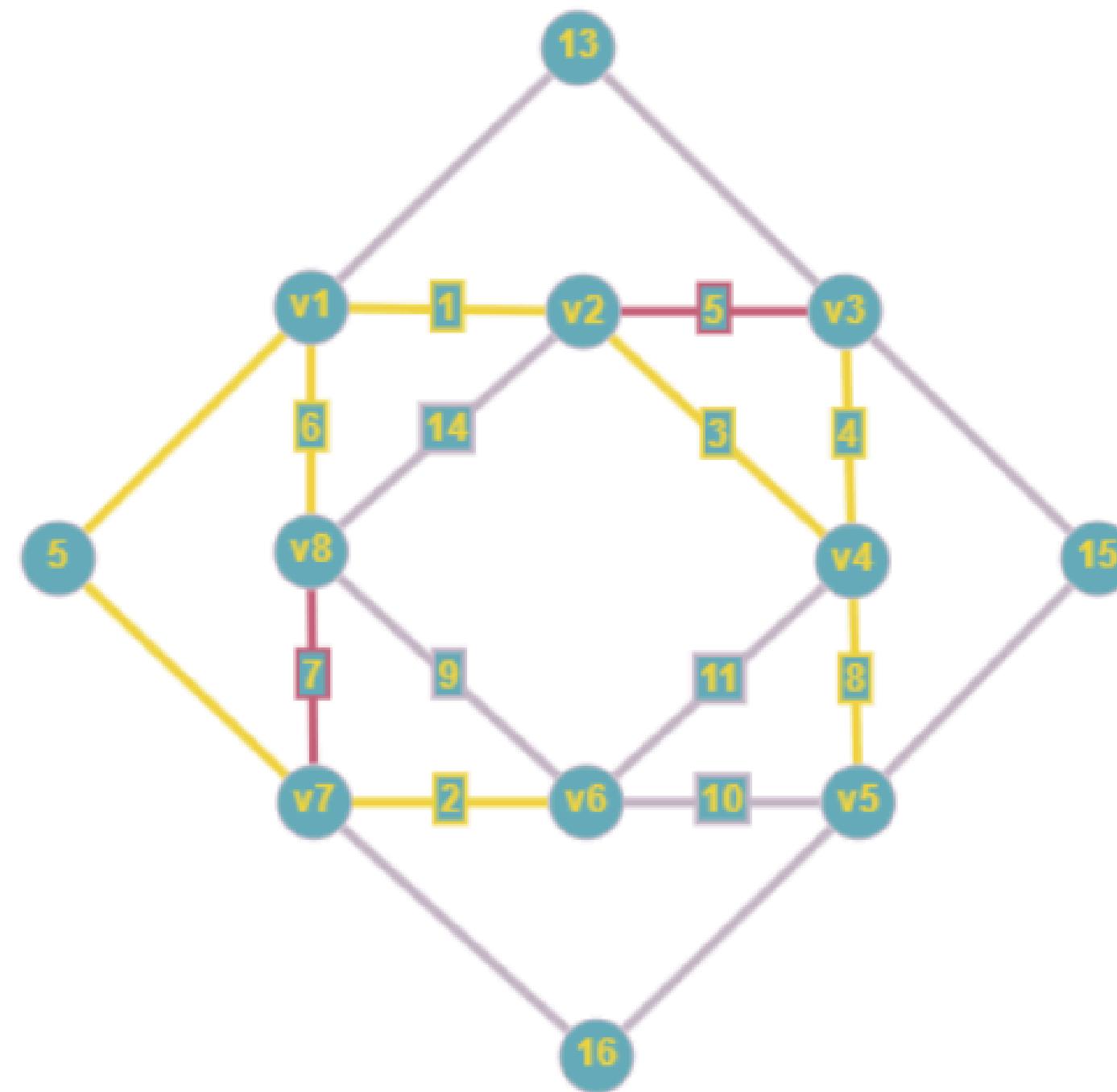
[8 (v4, v5)
9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

**Se toma el elemento
7 (v7, v8)**

No se modifica nada
[v1, v2, v3, v4, v6, v7, v8]

**v7 y v8 ya están en la lista, no es
agrega la arista**

El algoritmo de kruskal



Lista de aristas:

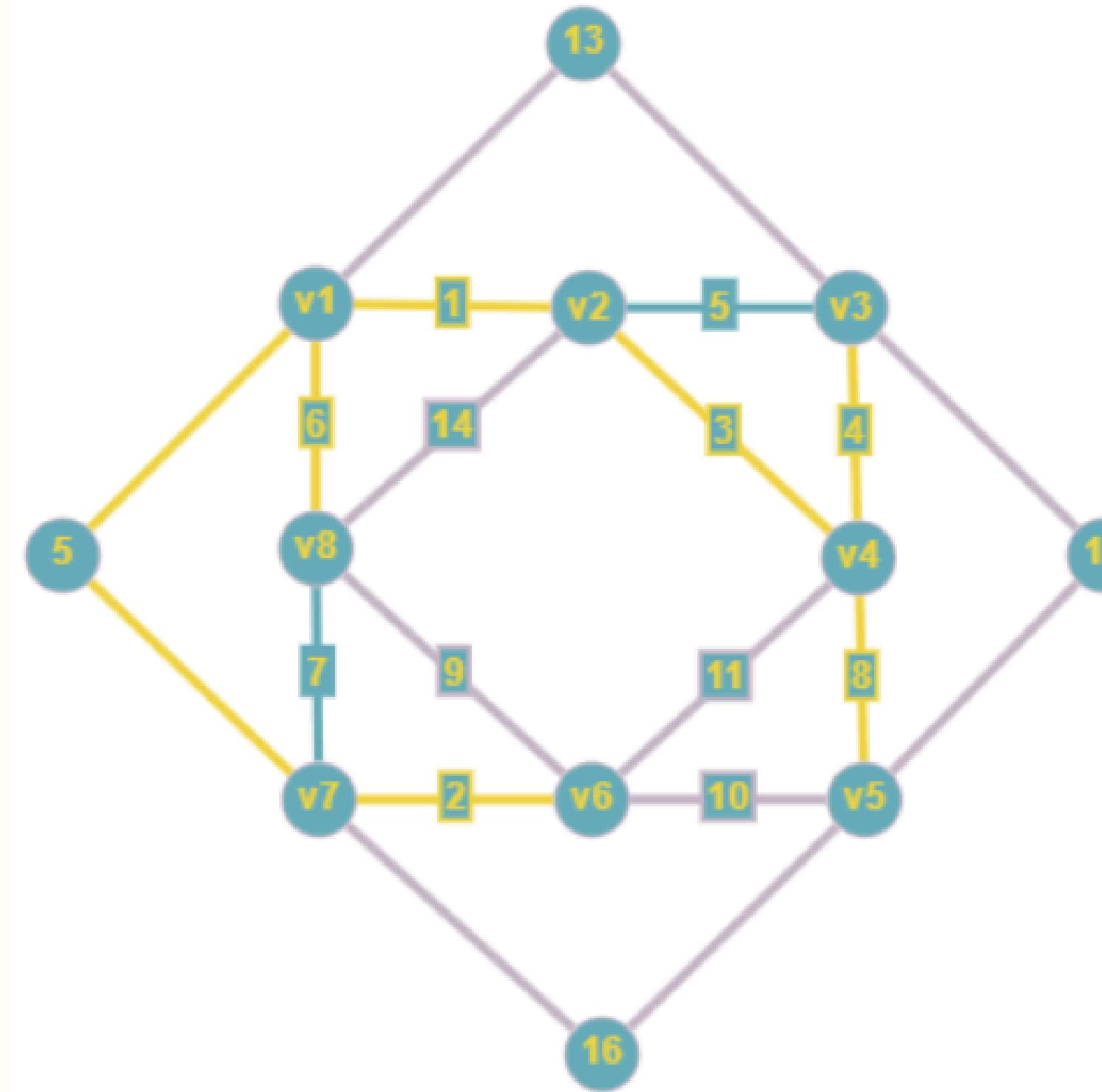
[9 (v6, v8)
10 (v5, v6)
11 (v4, v6)
13 (v1, v3)
14 (v2, v8)
15 (v3, v5)
16 (v5, v7)]

**Se toma el elemento
8 (v4, v5)**

**Se agrega v5
[v1, v2, v3, v4, v5, v6, v7, v8]**

Se agrega la arista

El algoritmo de kruskal



**Como solo hay una sola lista
conteniendo a los vértices, se
termina el proceso.**

El algoritmo de kruskal

Lista de adyacencia del grafo:

v1 → (v2, 5) → (v3, 1) → (v4, 6)

v2 → (v1, 5) → (v4, 2) → (v5, 3)

v3 → (v1, 1) → (v4, 10)

v4 → (v1, 6) → (v2, 2) → (v3, 10) → (v5, 2)

v5 → (v2, 3) → (v4, 2)

Lista de adyacencia del árbol:

Vacio

Lista de aristas:

[1 (v1, v3)

2 (v2, v4)

2 (v4, v5)

3 (v2, v5)

5 (v1, v2)

6 (v1, v4)

10 (v3, v4)]

El algoritmo de kruskal

Lista de adyacencia del grafo:

$v_1 \rightarrow (v_2, 5) \rightarrow (v_3, 1) \rightarrow (v_4, 6)$

$v_2 \rightarrow (v_1, 5) \rightarrow (v_4, 2) \rightarrow (v_5, 3)$

$v_3 \rightarrow (v_1, 1) \rightarrow (v_4, 10)$

$v_4 \rightarrow (v_1, 6) \rightarrow (v_2, 2) \rightarrow (v_3, 10) \rightarrow (v_5, 2)$

$v_5 \rightarrow (v_2, 3) \rightarrow (v_4, 2)$

Lista de aristas:

[2 (v_2, v_4)

2 (v_4, v_5)

3 (v_2, v_5)

5 (v_1, v_2)

6 (v_1, v_4)

10 (v_3, v_4)]

Se toma el elemento

1 (v_1, v_3)

Se crea la lista

[v_1, v_3]

Se agrega la arista

Lista de adyacencia del árbol:

$v_1 \rightarrow (v_3, 1)$

$v_3 \rightarrow (v_1, 1)$

El algoritmo de kruskal

Lista de adyacencia del grafo:

$v_1 \rightarrow (v_2, 5) \rightarrow (v_3, 1) \rightarrow (v_4, 6)$

$v_2 \rightarrow (v_1, 5) \rightarrow (v_4, 2) \rightarrow (v_5, 3)$

$v_3 \rightarrow (v_1, 1) \rightarrow (v_4, 10)$

$v_4 \rightarrow (v_1, 6) \rightarrow (v_2, 2) \rightarrow (v_3, 10) \rightarrow (v_5, 2)$

$v_5 \rightarrow (v_2, 3) \rightarrow (v_4, 2)$

Lista de aristas:

[2 (v_4, v_5)

3 (v_2, v_5)

5 (v_1, v_2)

6 (v_1, v_4)

10 (v_3, v_4)]

Se toma el elemento

2 (v_2, v_4)

Se crea la lista

[v_2, v_4]

[v_1, v_3]

Lista de adyacencia del árbol:

$v_1 \rightarrow (v_3, 1)$

$v_3 \rightarrow (v_1, 1)$

$v_2 \rightarrow (v_4, 2)$

$v_4 \rightarrow (v_2, 2)$

Se agrega la arista

El algoritmo de kruskal

Lista de adyacencia del grafo:

$v_1 \rightarrow (v_2, 5) \rightarrow (v_3, 1) \rightarrow (v_4, 6)$

$v_2 \rightarrow (v_1, 5) \rightarrow (v_4, 2) \rightarrow (v_5, 3)$

$v_3 \rightarrow (v_1, 1) \rightarrow (v_4, 10)$

$v_4 \rightarrow (v_1, 6) \rightarrow (v_2, 2) \rightarrow (v_3, 10) \rightarrow (v_5, 2)$

$v_5 \rightarrow (v_2, 3) \rightarrow (v_4, 2)$

Lista de aristas:

[5 (v_1, v_2)

6 (v_1, v_4)

10 (v_3, v_4)]

Se toma el elemento

3 (v_2, v_5)

No se modifica nada

[v_2, v_4, v_5]

[v_1, v_3]

Lista de adyacencia del árbol:

$v_1 \rightarrow (v_3, 1)$

$v_3 \rightarrow (v_1, 1)$

$v_2 \rightarrow (v_4, 2)$

$v_4 \rightarrow (v_2, 2) \rightarrow (v_5, 2)$

$v_5 \rightarrow (v_4, 2)$

Como v_2 y v_5 están en la misma lista, no se agrega la arista

El algoritmo de kruskal

Lista de adyacencia del grafo:

$v_1 \rightarrow (v_2, 5) \rightarrow (v_3, 1) \rightarrow (v_4, 6)$

$v_2 \rightarrow (v_1, 5) \rightarrow (v_4, 2) \rightarrow (v_5, 3)$

$v_3 \rightarrow (v_1, 1) \rightarrow (v_4, 10)$

$v_4 \rightarrow (v_1, 6) \rightarrow (v_2, 2) \rightarrow (v_3, 10) \rightarrow (v_5, 2)$

$v_5 \rightarrow (v_2, 3) \rightarrow (v_4, 2)$

Lista de aristas:

[6 (v_1, v_4)

10 (v_3, v_4)]

Se toma el elemento

5 (v_1, v_2)

Se juntan las listas

[v_1, v_2, v_3, v_4, v_5]

Se agrega la arista

Lista de adyacencia del árbol:

$v_1 \rightarrow (v_3, 1) \rightarrow (v_2, 5)$

$v_3 \rightarrow (v_1, 1)$

$v_2 \rightarrow (v_1, 5) \rightarrow (v_4, 2)$

$v_4 \rightarrow (v_2, 2) \rightarrow (v_5, 2)$

$v_5 \rightarrow (v_4, 2)$

El algoritmo de kruskal

Lista de adyacencia del árbol:

v1 → (v3, 1) → (v2, 5)

v3 → (v1, 1)

v2 → (v1, 5) → (v4, 2)

v4 → (v2, 2) → (v5, 2)

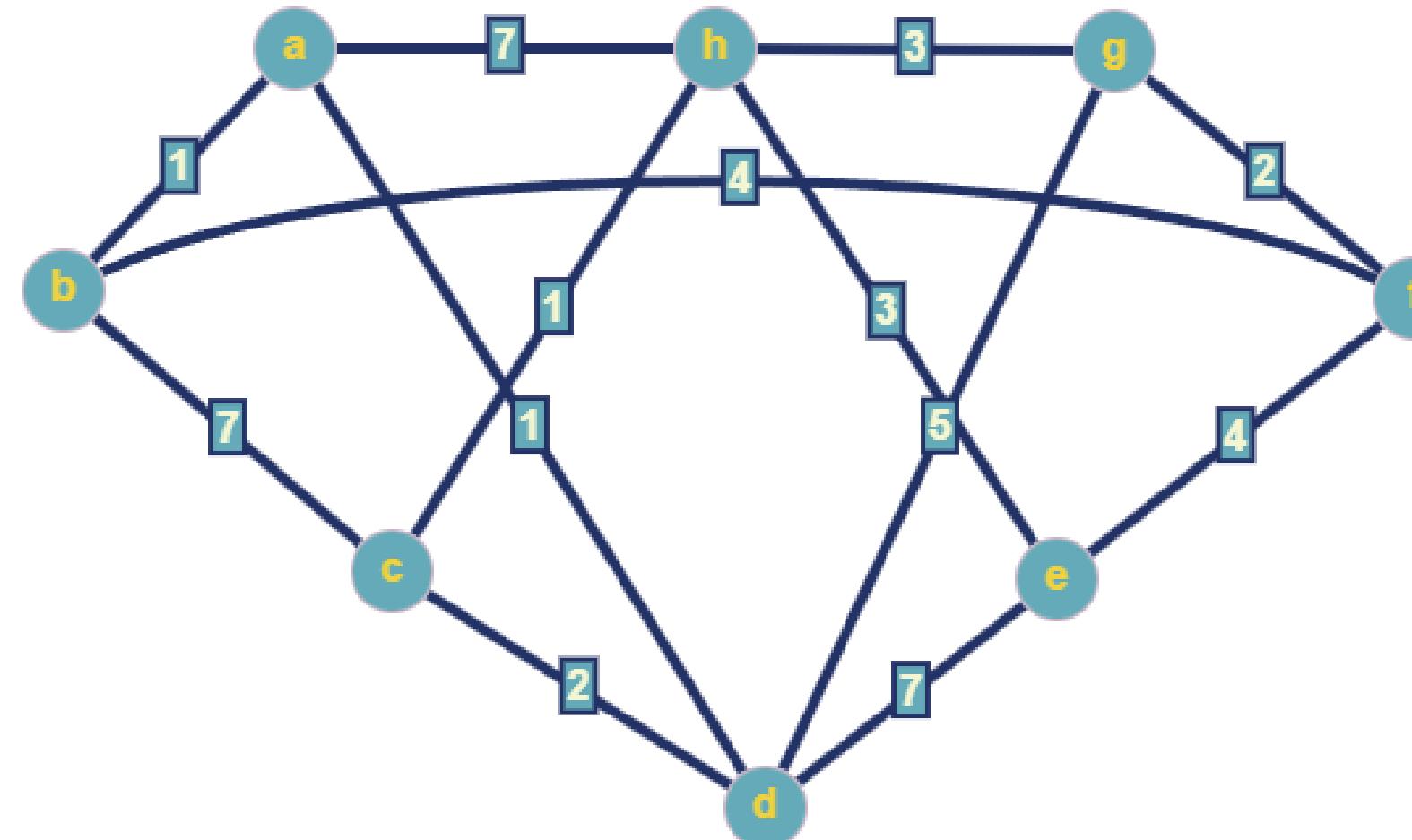
v5 → (v4, 2)

Como todas las aristas se encuentran en la misma lista, se acaba el algoritmo

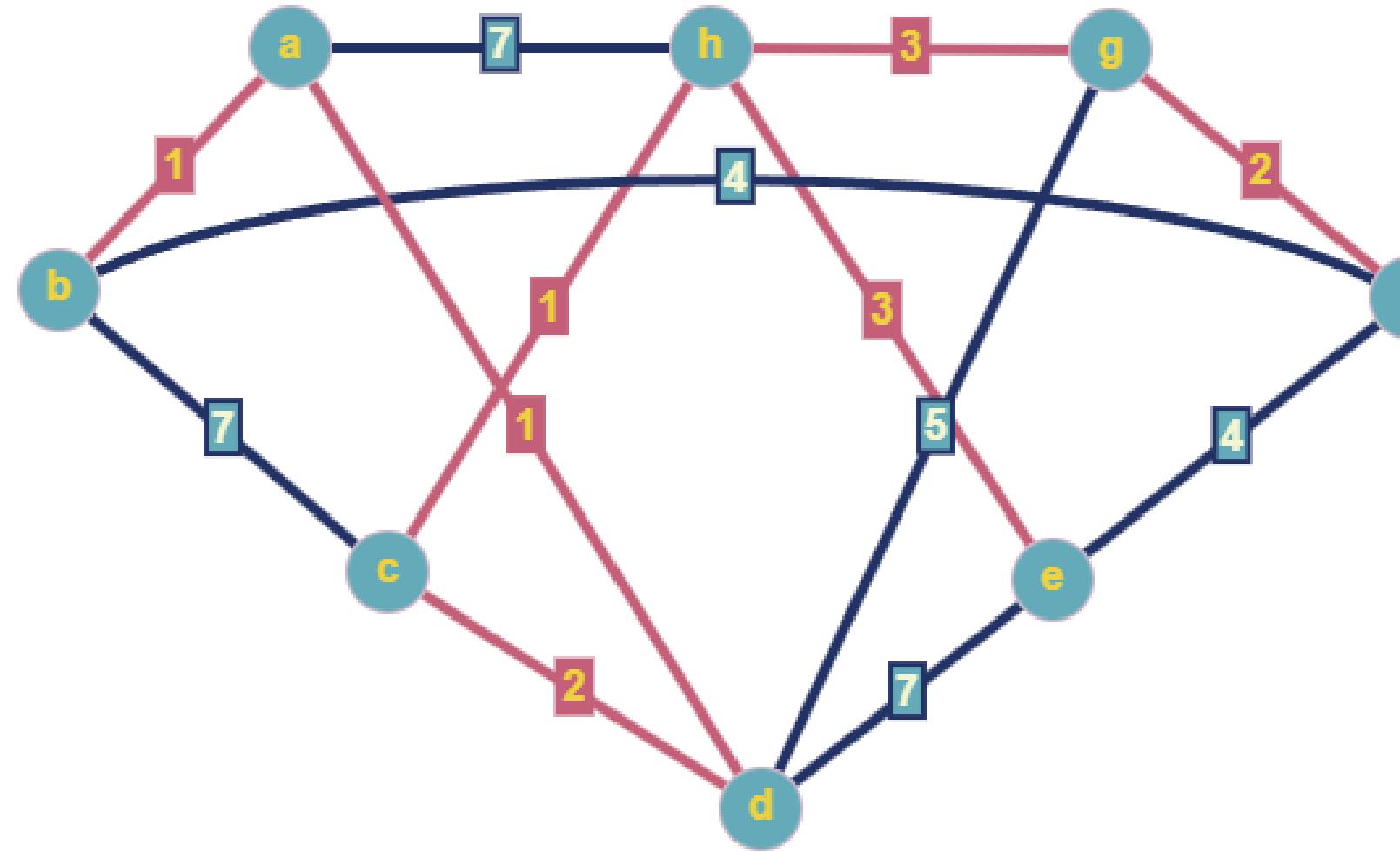
EJERCICIOS

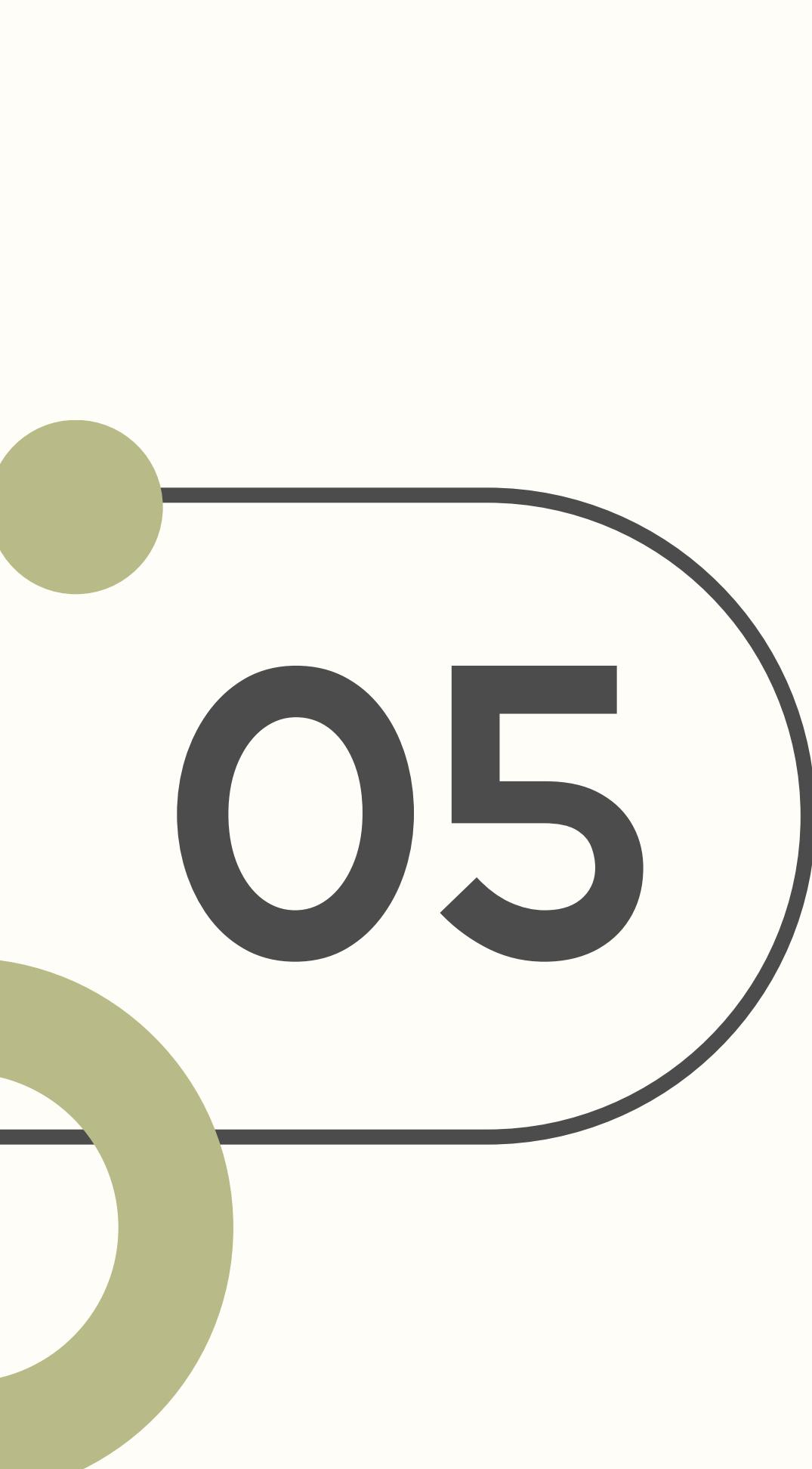
Ejercicio

Obtener el árbol de expansión mínima del siguiente grafo iniciando desde el vértice “e” con el algoritmo de Prim y obtenerlo también con Kruskal.



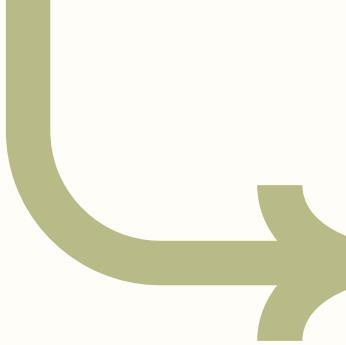
Solucion





05

Métodos de búsqueda

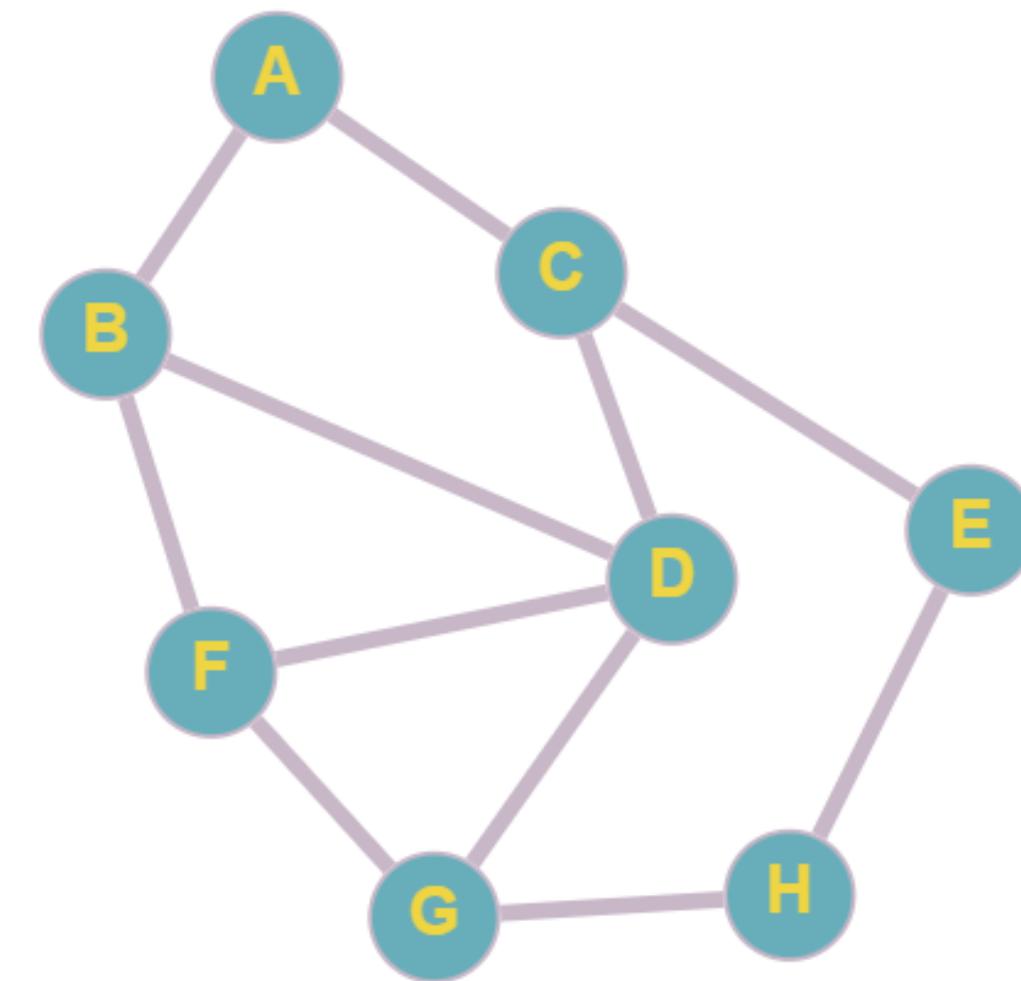


Busqueda

Se puede aplicar la búsqueda con el mismo algoritmo con los que recorremos los grafos (Anchura y Profundidad), nada más que cuando encontramos nuestro elemento deseado, devolvemos el camino para llegar a él.

Anchura

Se desea encontrar el nodo ‘D’ realizando un recorrido por anchura comenzado desde el nodo ‘A’ del siguiente grafo:

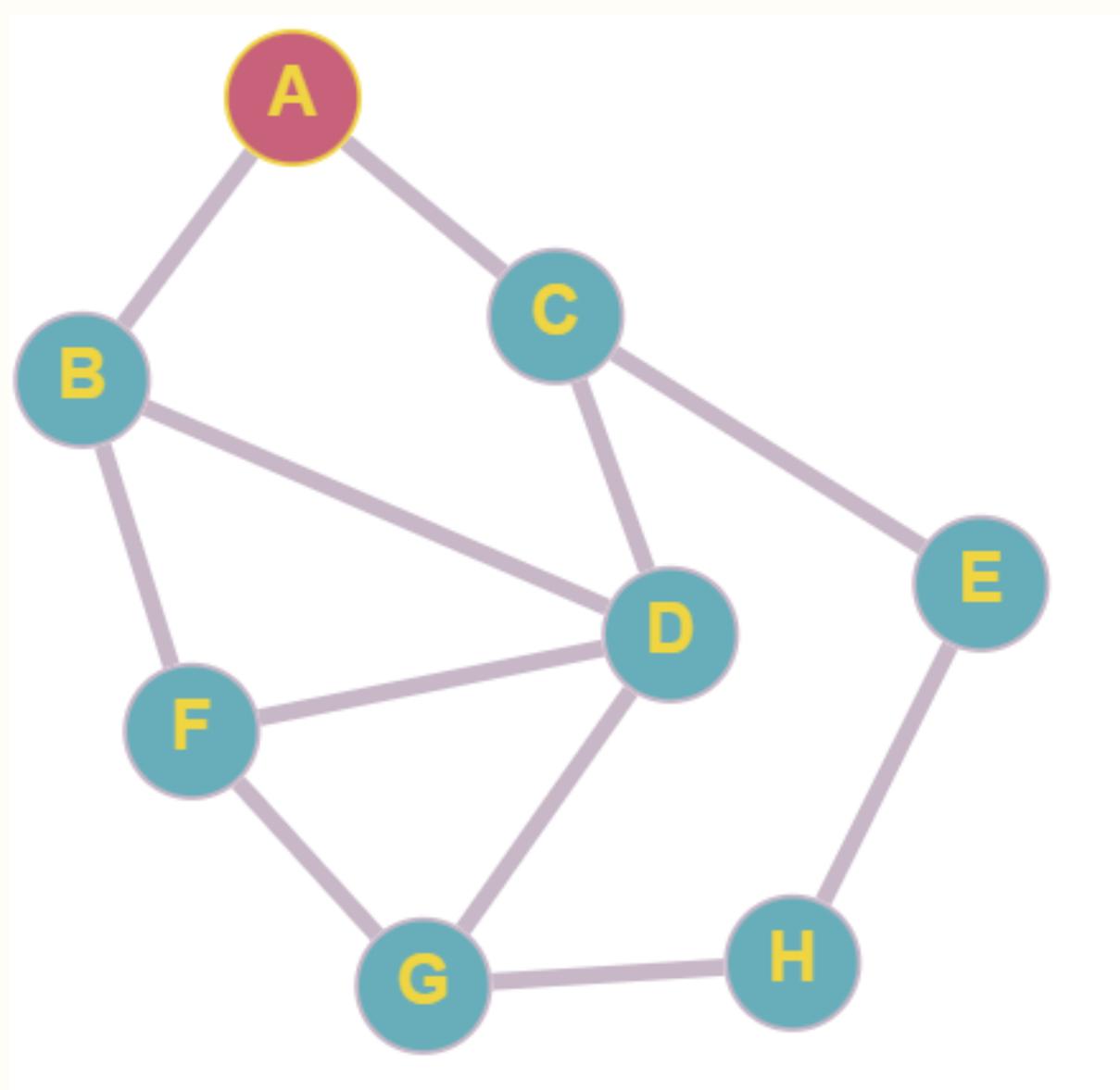


$G(V, A)$

$V: \{ A, B, C, D, E, F, G, H \}$

$A: \{ (A, B), (A, C), (B, F), (B, D), (C, A), (C, D), (C, E), (D, B), (D, C), (D, F), (D, G), (E, C), (E, H), (F, B), (F, D), (F, G), (G, D), (G, F), (G, H), (H, E), (H, F) \}$

Anchura



Paso 1: Inicialización

Comenzamos en el nodo A.

Se encola A → Cola: [A]

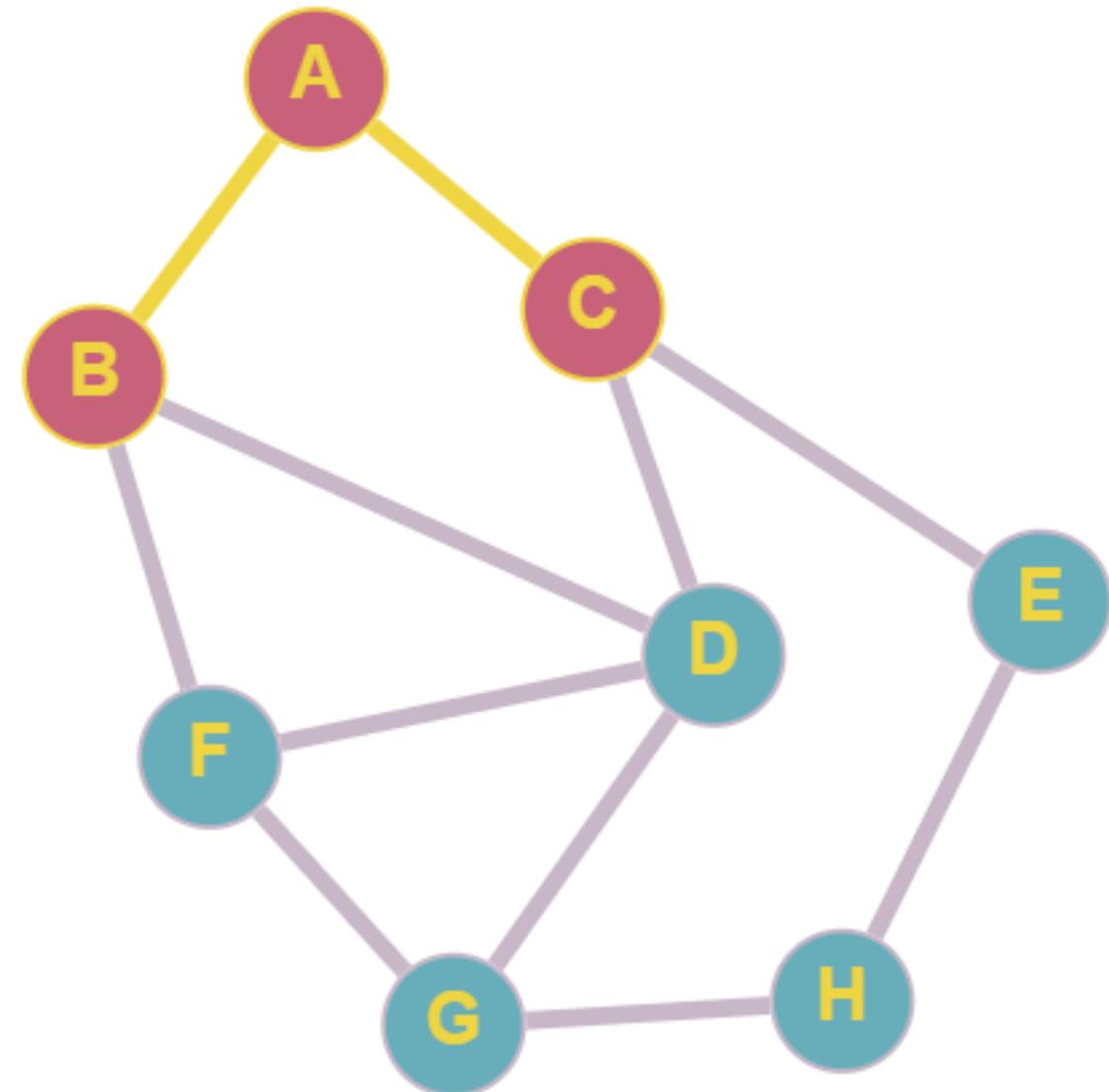
Se marca A como visitado.

Lista de visitados:

{A}

V	P(V)
A	NULL
B	NULL
C	NULL
D	NULL
E	NULL
F	NULL
G	NULL
H	NULL

Anchura

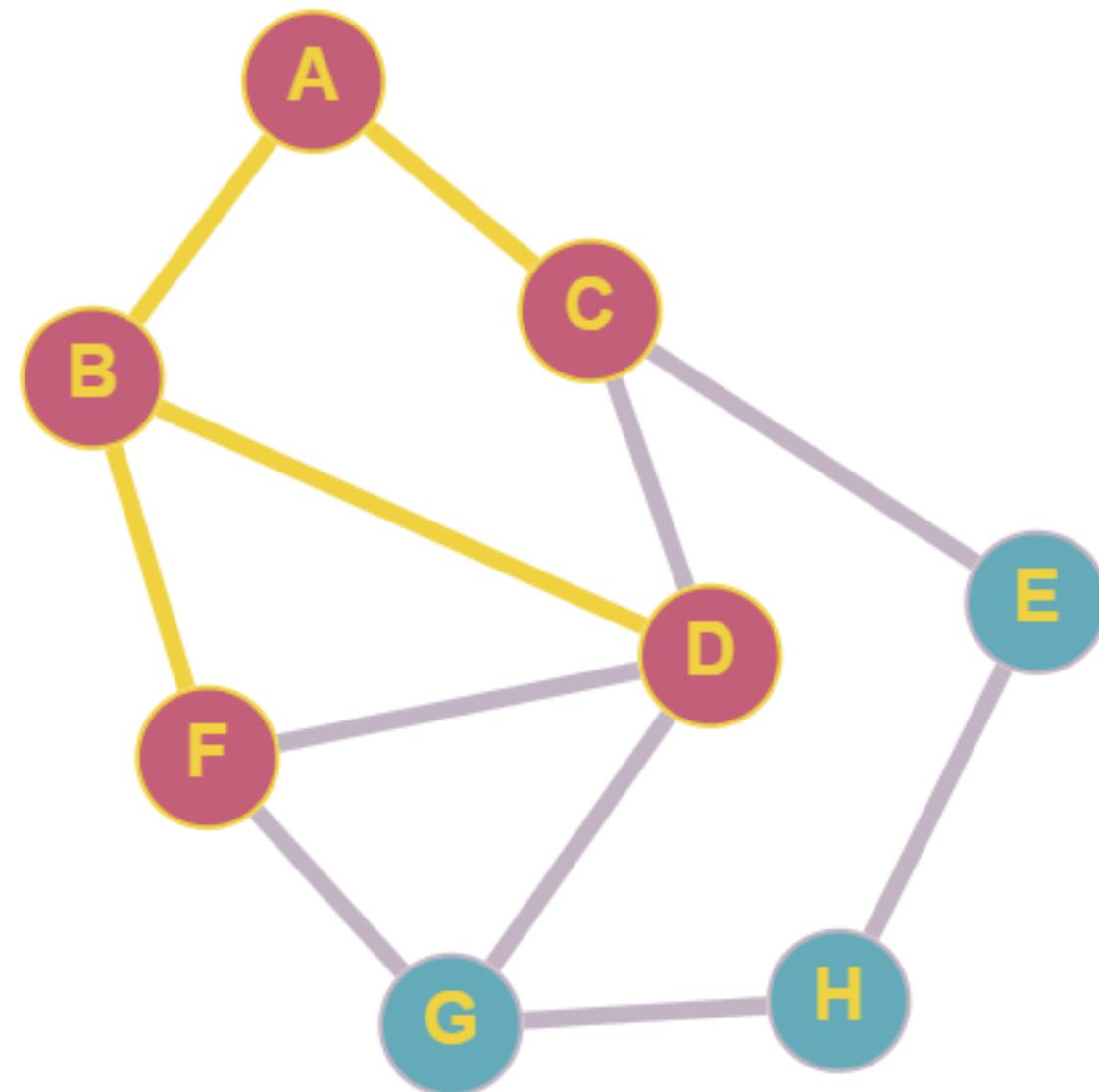


Paso 2: Procesamos el nodo A
Desencolamos A → Cola: [].
Tiene como nodos vecinos no visitados B y C
Se encolan B, C → Cola: [B,C]
Se marcan B, C como visitado.
Lista de visitados:
{A,B,C}

Recorrido actual:
A

V	P(V)
A	NULL
B	A
C	A
D	NULL
E	NULL
F	NULL
G	NULL
H	NULL

Anchura



Paso 3: Procesamos el nodo B

Desencolamos B → Cola: [C].

Tiene como nodos vecinos no visitados

D y F

Se encola D, F → Cola: [C,D,F]

Se marcan D,F como visitado.

Lista de visitados:

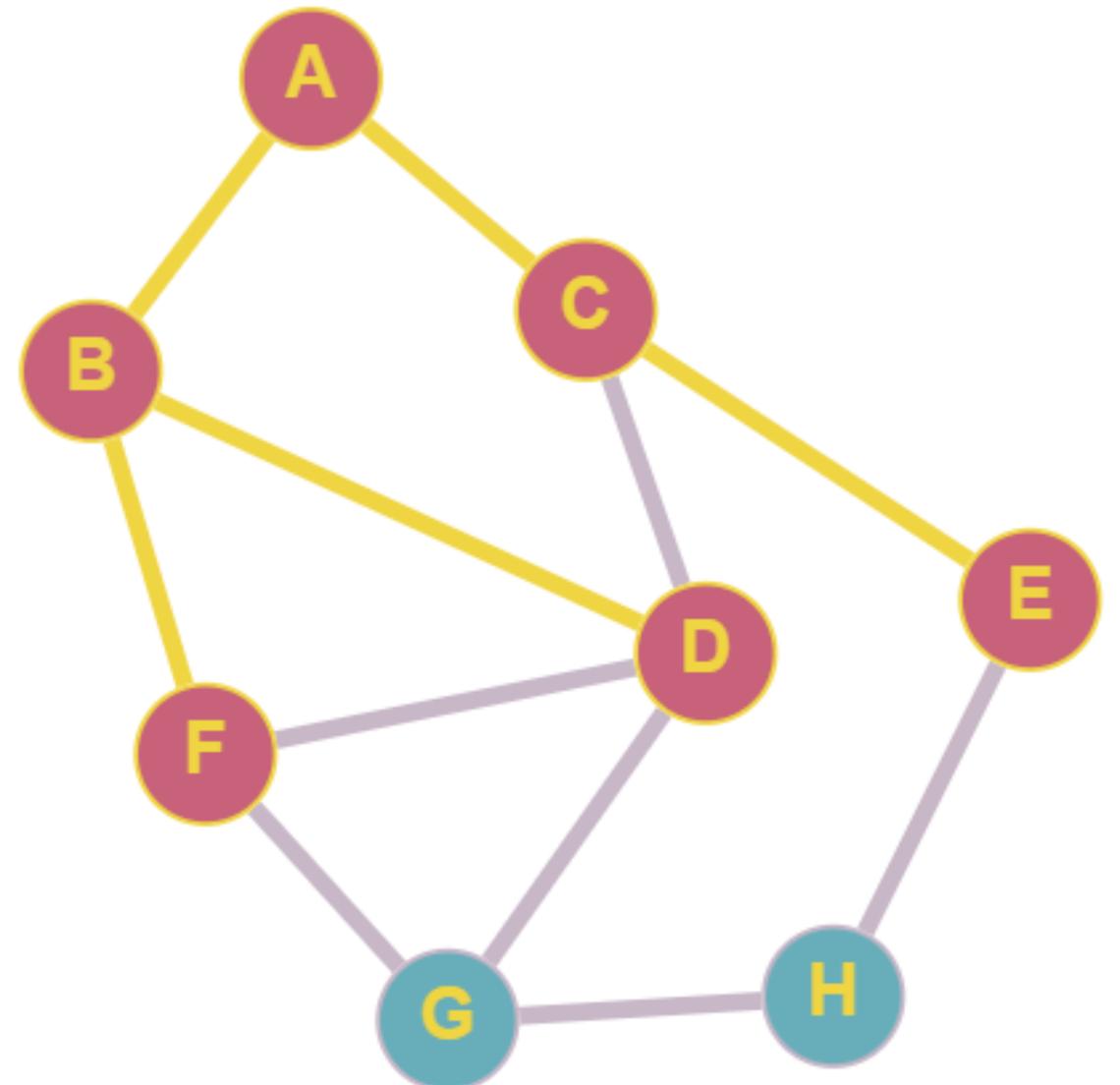
{A,B,C,D,F}

Recorrido actual:

A → B

V	P(V)
A	NULL
B	A
C	A
D	B
E	NULL
F	B
G	NULL
H	NULL

Anchura

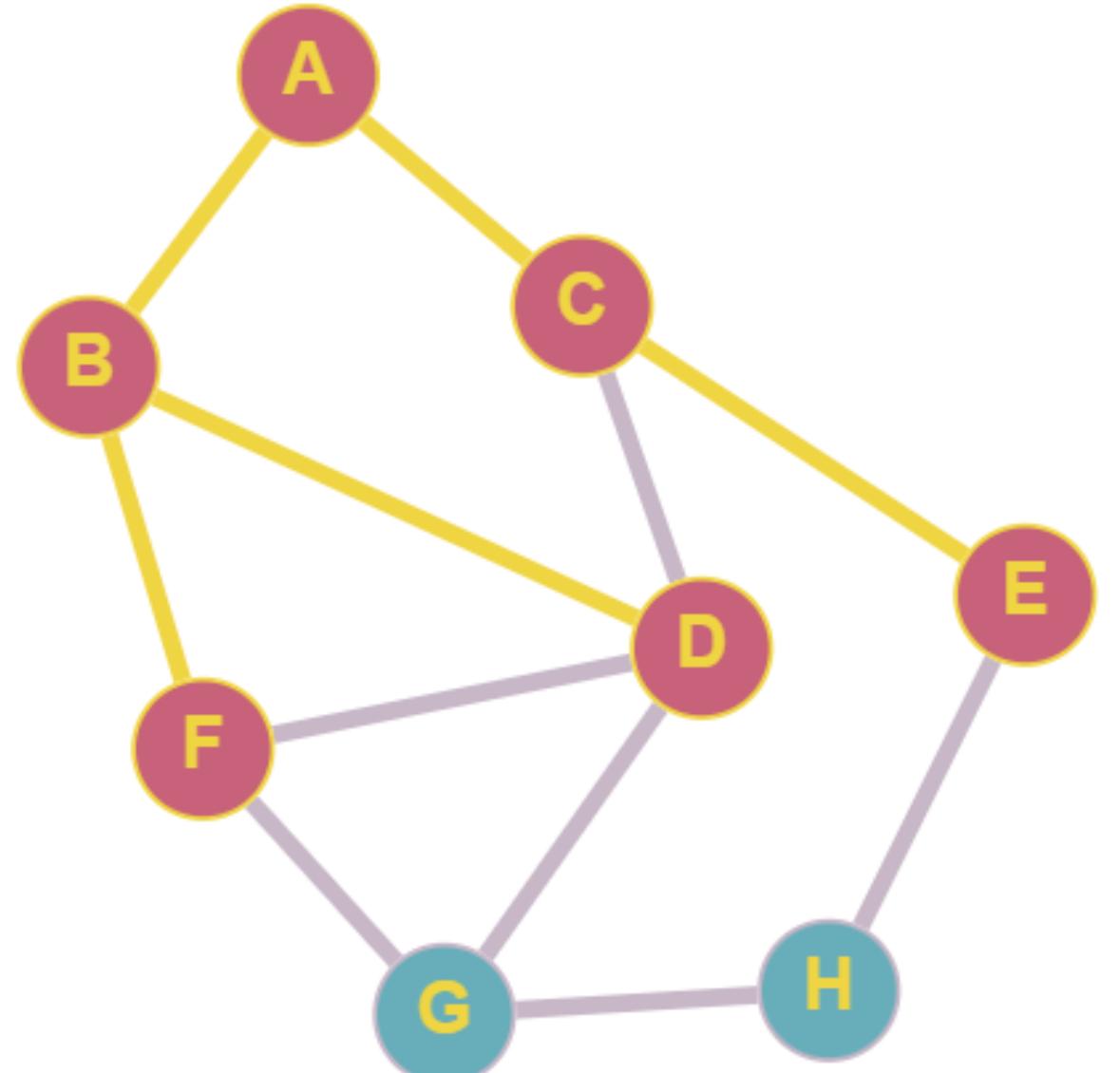


Paso 4: Procesamos el nodo C
Desencolamos C → Cola: [D,F].
Tiene como nodo vecino no visitado E
Se encola E → Cola: [D,F,E]
Se marcan E como visitado.
Lista de visitados:
{A,B,C,D,F,E}

Recorrido actual:
A → B → C

V	P(V)
A	NULL
B	A
C	A
D	B
E	C
F	B
G	NULL
H	NULL

Anchura



Paso 5: Procesamos el nodo D
Desencolamos $D \rightarrow$ Cola: [F,E].
Como D es el nodo que buscamos, entonces paramos y realizamos el corrido usando la tabla.

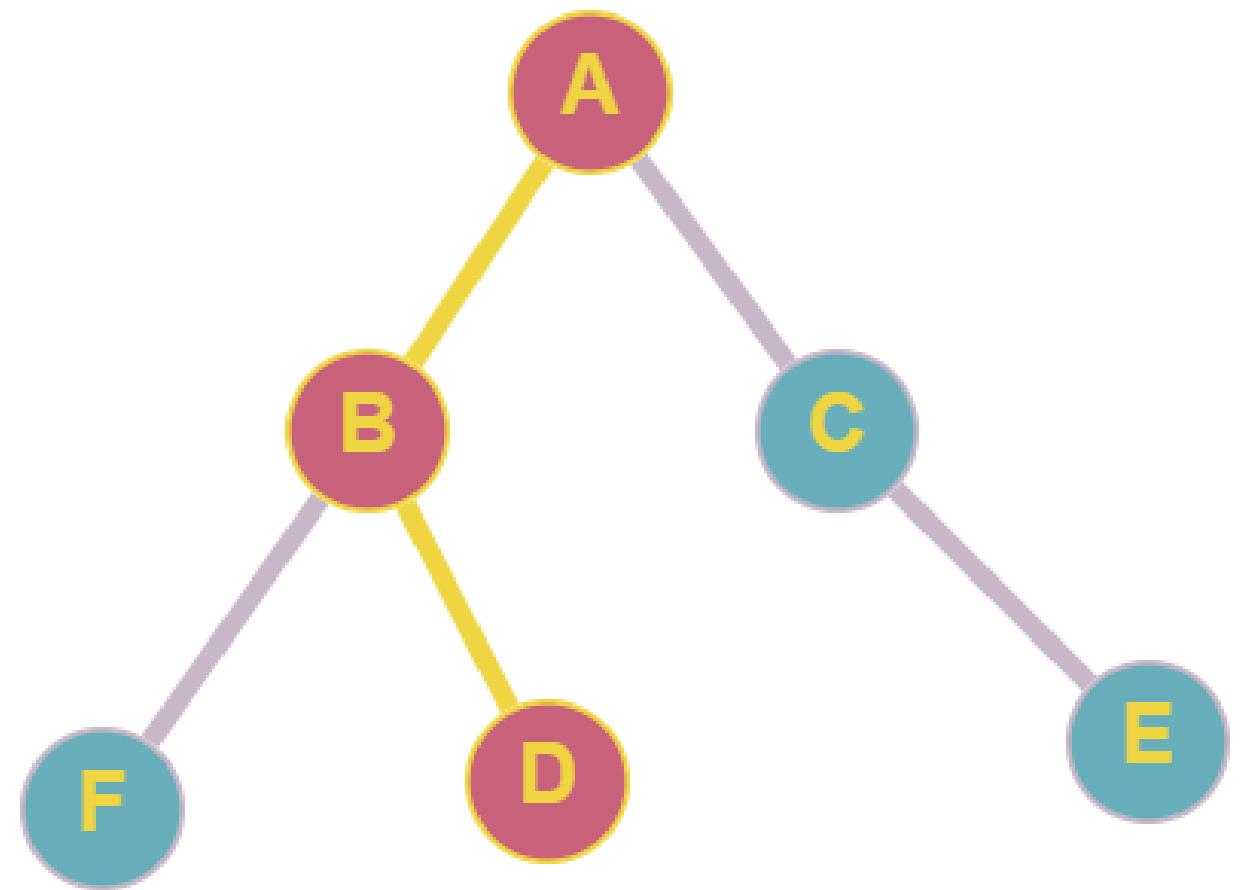
Lista de visitados:

{A,B,C,D,F,E}

Recorrido actual:
 $A \rightarrow B \rightarrow C \rightarrow D$

V	P(V)
A	NULL
B	A
C	A
D	B
E	C
F	B
G	D
H	NULL

Anchura



Entonces usando la tabla tenemos que el recorrido para el nodo D es:

$A \rightarrow B \rightarrow D$

V	P(V)
A	NULL
B	A
C	A
D	B
E	C
F	B
G	D
H	NULL

Profundidad

M	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	0	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	1	1	0	0	1
F	0	0	0	0	1	0

Paso 1: Inicialización

Comenzamos en el nodo A.

Se coloca en la pila: A → Pila: [A]

Se marca A como visitado.

Lista de visitados:

{A}

V	P(V)
A	NULL
B	NULL
C	NULL
D	NULL
E	NULL
F	NULL

M

Profundidad

M	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	0	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	1	1	0	0	1
F	0	0	0	0	1	0

Paso 2: Procesamos A

Sacamos de la pila A → Pila: [].

Tiene como nodo vecino no visitado el B

Se coloca en la pila: B → Pila: [B]

Se marca B como visitado.

Lista de visitados:

{A,B}

Recorrido actual:

A

M	A
A	0

V	P(V)
A	NULL
B	A
C	NULL
D	NULL
E	NULL
F	NULL

Profundidad

M	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	0	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	1	1	0	0	1
F	0	0	0	0	1	0

Paso 3: Procesamos B

Sacamos de la pila B → Pila: [].

Tiene como nodos vecinos no visitados el D y E

Se coloca en la pila: D, E → Pila: [E,D]

Se marca D y E como visitado.

Lista de visitados:

{A,B,D,E}

Recorrido actual:

A → B

M	A	B
A	0	1
B	1	0

V	P(V)
A	NULL
B	A
C	NULL
D	B
E	B
F	NULL

Profundidad

M	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	0	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	1	1	0	0	1
F	0	0	0	0	1	0

Paso 4: Procesamos E

Sacamos de la pila E → Pila: [D].

Tiene como nodos vecinos no visitados el C y F.

Se coloca en la pila: C, F → Pila: [F,C,D]

Se marca F y C como visitado.

Lista de visitados:

{A,B,D,E,F,C}

Recorrido actual:

A → B → E

M	A	B	E
A	0	1	0
B	1	0	1
E	0	1	0

V	P(V)
A	NULL
B	A
C	E
D	B
E	B
F	E

Profundidad

M	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	0	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	1	1	0	0	1
F	0	0	0	0	1	0

M	A	B	E	F
A	0	1	0	0
B	1	0	1	0
E	0	1	0	1
F	0	0	1	0

Paso 5: Procesamos F
 Sacamos de la pila F → Pila:
 [C,D].
 No tiene nodos no visitados.
 No apilamos nada → Pila: [C,D]

Lista de visitados:

{A,B,D,E,F,C}

Recorrido actual:
 A → B → E → F

V	P(V)
A	NULL
B	A
C	E
D	B
E	B
F	E

Profundidad

M	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	0	1	1	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	1	1	0	0	1
F	0	0	0	0	1	0

Paso 6: Procesamos C
Sacamos de la pila C → Pila: [D].
Como C es el nodo buscado paramos el recorrido.

Lista de visitados:
{A,B,D,E,F,C}

M	A	B	E	F	C
A	0	1	0	0	0
B	1	0	1	0	0
E	0	1	0	1	1
F	0	0	1	0	0
C	0	0	1	0	0

Recorrido actual:
A → B → E → F → C

V	P(V)
A	NULL
B	A
C	E
D	B
E	B
F	E

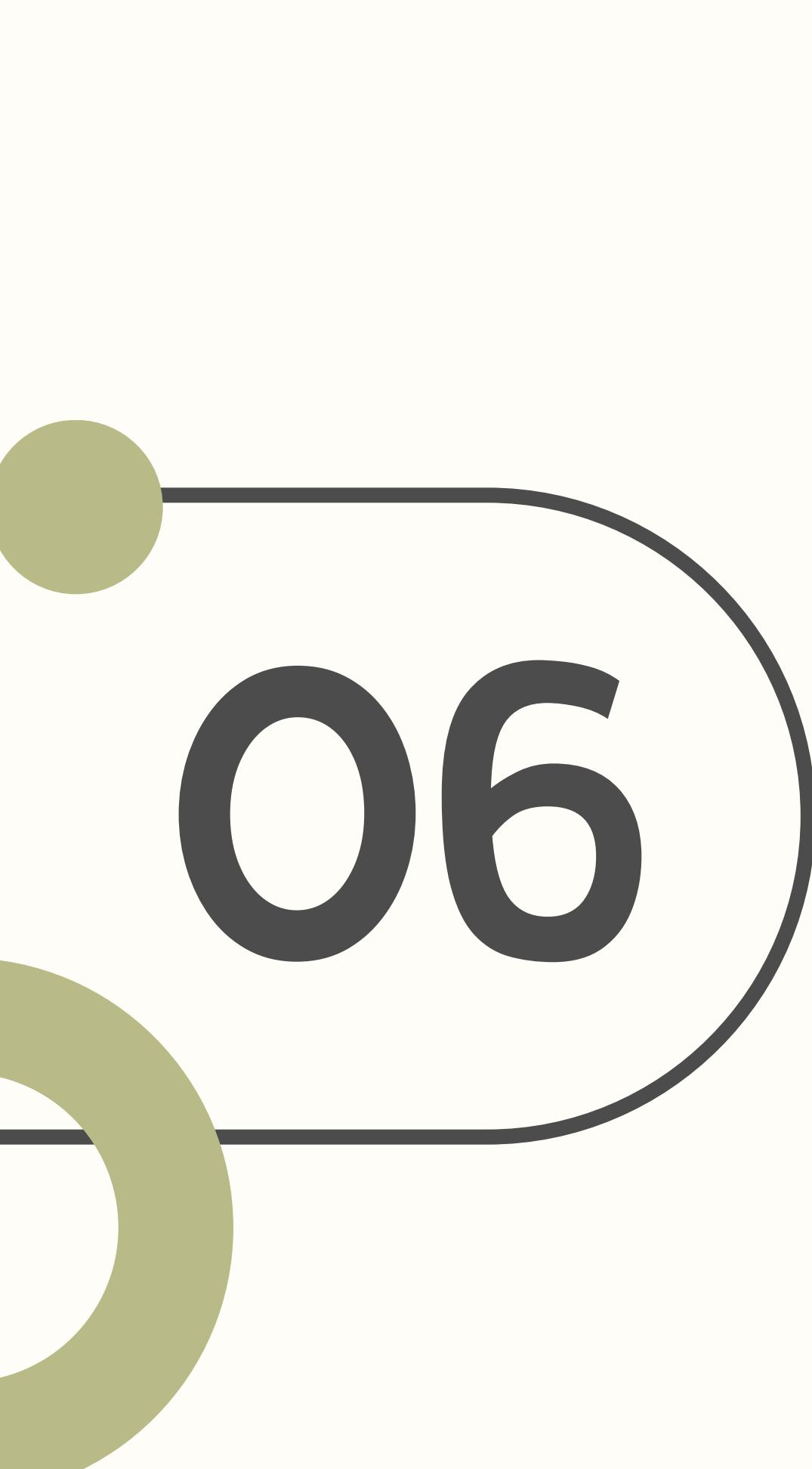
Profundidad

M	A	B	E	F	C
A	0	1	0	0	0
B	1	0	1	0	0
E	0	1	0	1	1
F	0	0	1	0	0
C	0	0	1	0	0

Entonces, usando la tabla tenemos que el recorrido para el nodo C es:

$$A \rightarrow B \rightarrow E \rightarrow C$$

V	P(V)
A	NULL
B	A
C	E
D	B
E	B
F	E



06

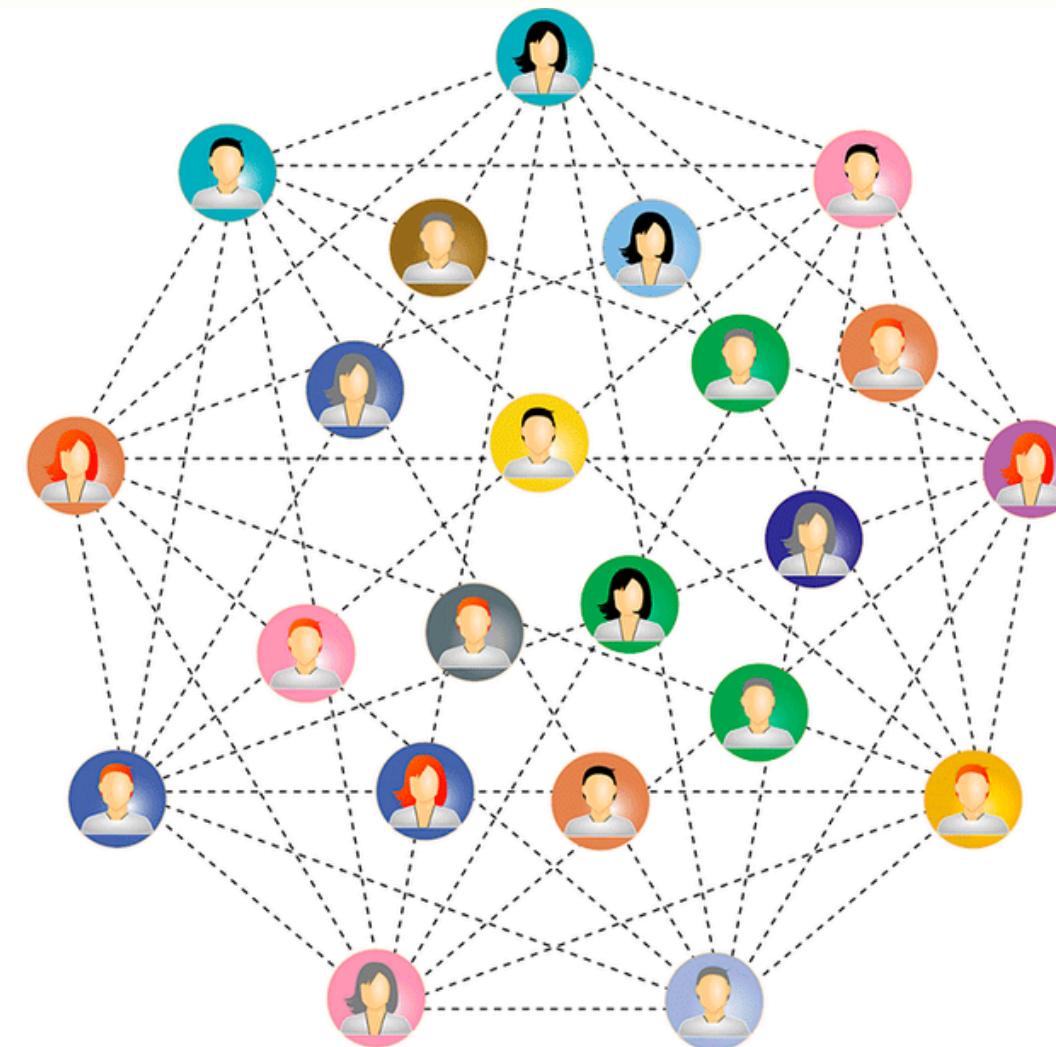
Aplicaciones

Aplicaciones

Diseño de redes

Los grafos ayudan a diseñar y resolver problemas, minimizando costos y maximizando la eficiencia.

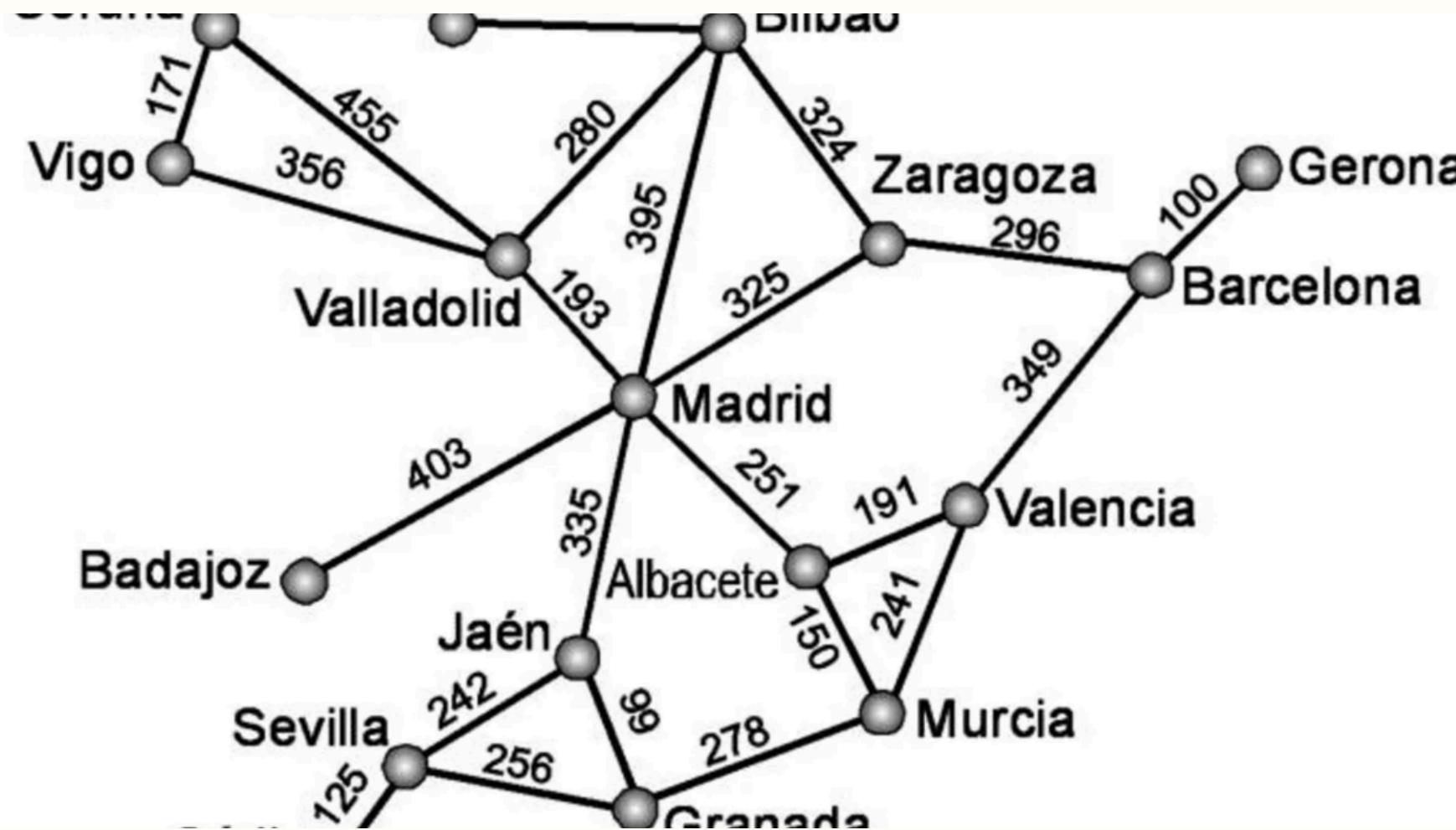
**Por ejemplo para telecomunicaciones:
Para conectar varias ciudades con fibra óptica, se busca la manera de interconectarlas con la menor cantidad de cable posible, reduciendo costos sin comprometer la conectividad.**



Aplicaciones

Diseño de rutas de transporte.

Los grafos son esenciales en la planificación de redes de transporte, como carreteras, vías ferroviarias, optimizando las conexiones y reduciendo costos.



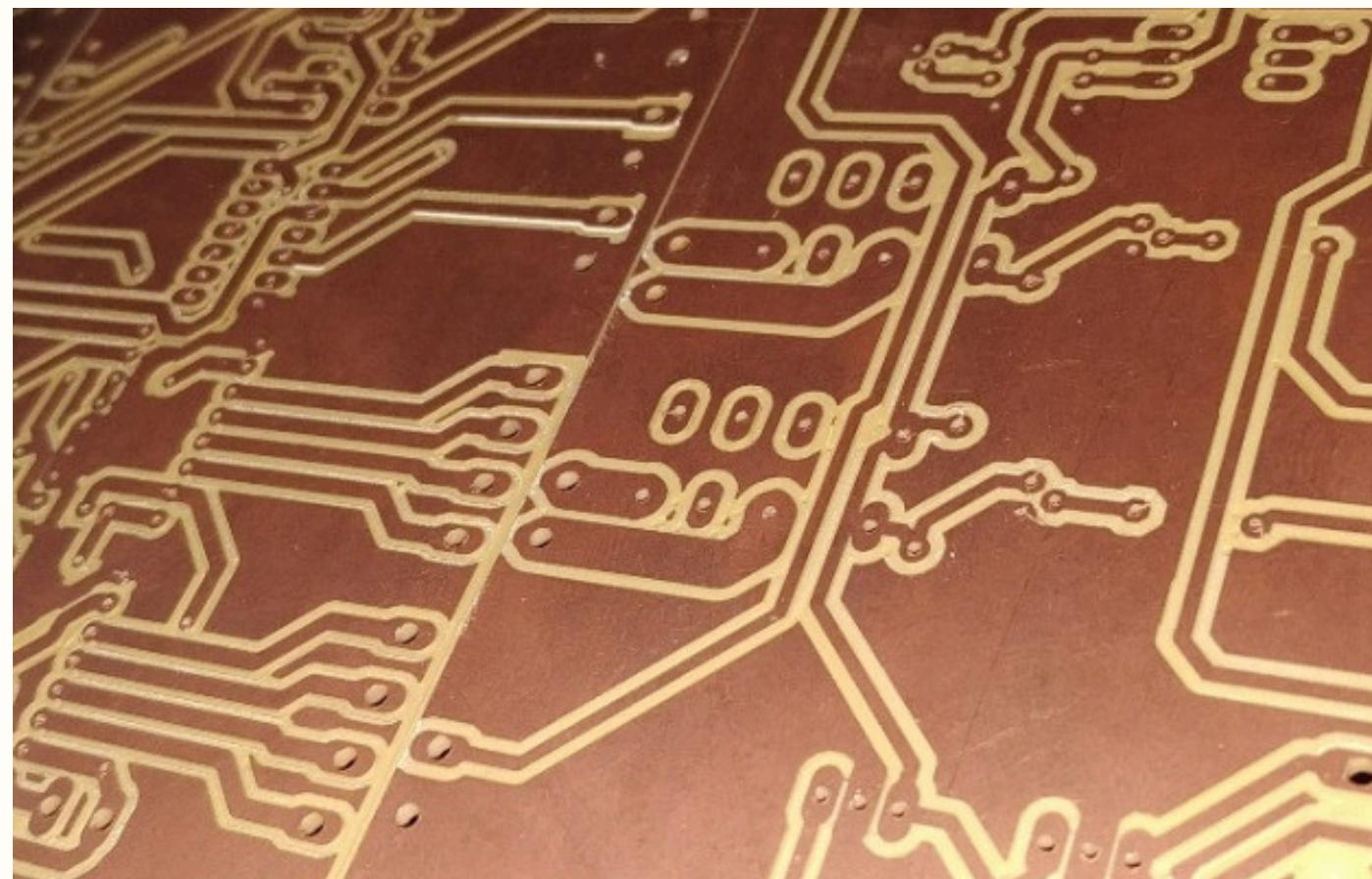
Planificación urbana y redes de agua o gas

Los grafos permiten modelar sistemas de distribución de recursos, asegurando la conectividad entre edificios y minimizando costos de construcción.

Aplicaciones

Diseño de circuitos impresos.

En cuanto al área de la fabricación de placas electrónicas, los grafos ayudan a planificar la mejor manera de conectar los componentes con la menor cantidad de pistas conductoras.





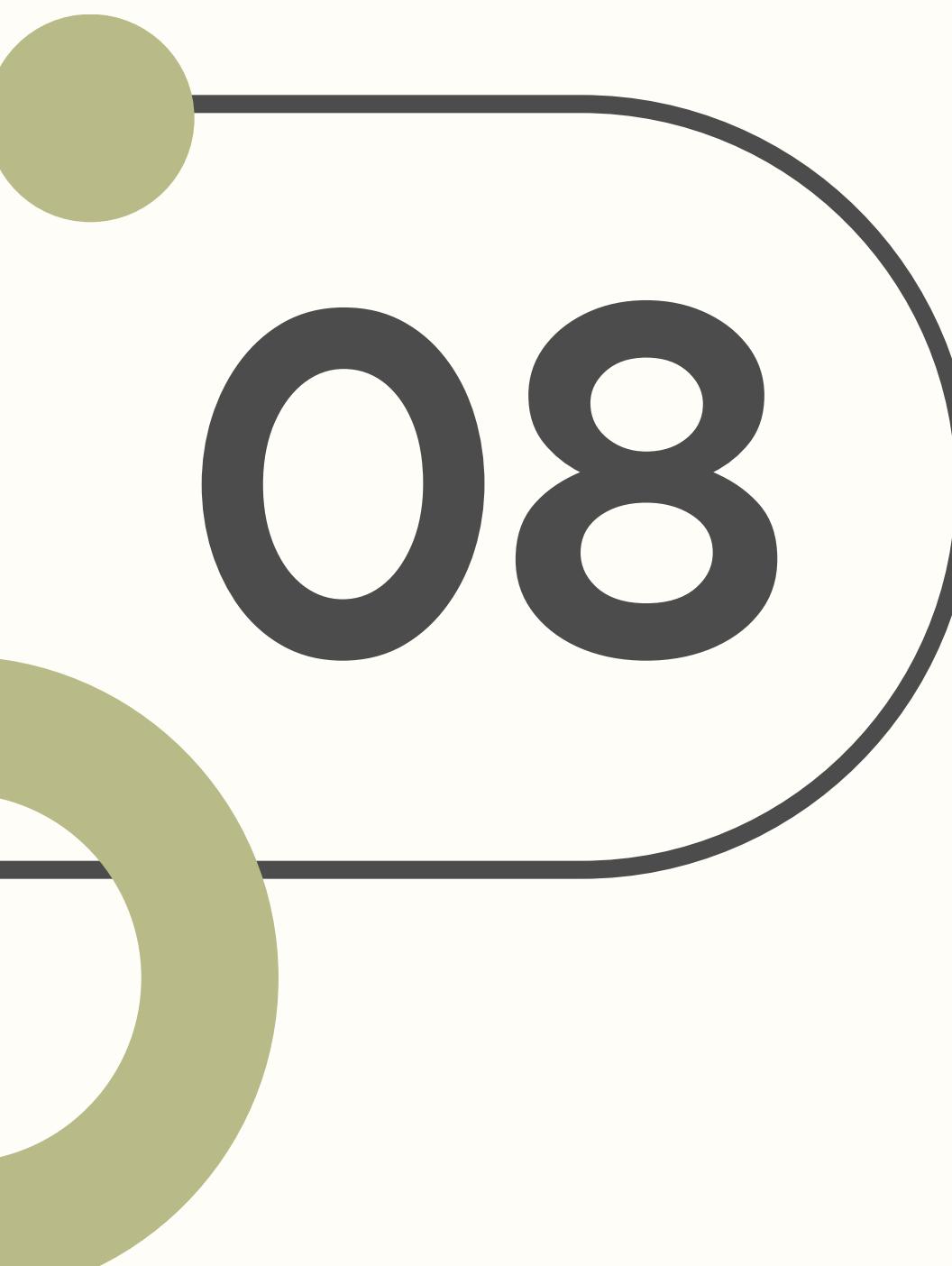
07

Conclusión

Conclusion

Los grafos son una herramienta matemática esencial para representar y analizar relaciones entre objetos en diversos sistemas complejos del mundo real. Con una estructura basada en nodos y aristas, así como gracias a sus algoritmos, permiten modelar redes sociales, mapas de rutas y navegación, conexiones eléctricas o de internet, procesos computacionales, flujos de información, además de problemas de optimización de recurso. Consideramos que gracias a los grafos son fundamentales para comprender, optimizar y resolver problemas en áreas como la informática, ingeniería, matemáticas, logística e inteligencia artificial, entre muchas otras.





08

Bibliografias

Bibliografías

GraphEverywhere, E. (2020, 10 marzo). Qué son los grafos. GraphEverywhere.

<https://www.grapheverywhere.com/que-son-los-grafos/>

What is Graph Traversal and Its Algorithms. (s. f.). Dgraph Blog.

<https://dgraph.io/blog/post/graph-traversal-algorithms/>

Mate. (2024, 14 octubre). Teoría de grafos: Conceptos clave y resultados impactantes.

Matematix. https://matematix.org/que-es-la-teoria-de-grafos/#Grafos_no_dirigidos

GraphEverywhere, E. (s. f.). Grafos I Qué son, tipos, orden y herramientas de visualización. GraphEverywhere. <https://www.grapheverywhere.com/grafos-que-son-tipos-orden-y-herramientas-de-visualizacion/>

GraphEverywhere, E. (2020b, marzo 10). Qué son los grafos. GraphEverywhere.

<https://www.grapheverywhere.com/que-son-los-grafos/>

La utilidad y aplicación de los Grafos y Sistemas de Información Geográfica. (s. f.).

Delfino.cr. <https://delfino.cr/2023/01/la-utilidad-y-aplicacion-de-los-grafos-y-sistemas-de-informacion-geografica>

Cairó Battistutti, O., & Guardati, S. (2006). Estructuras de datos (3.^a ed.). McGraw Hill.