

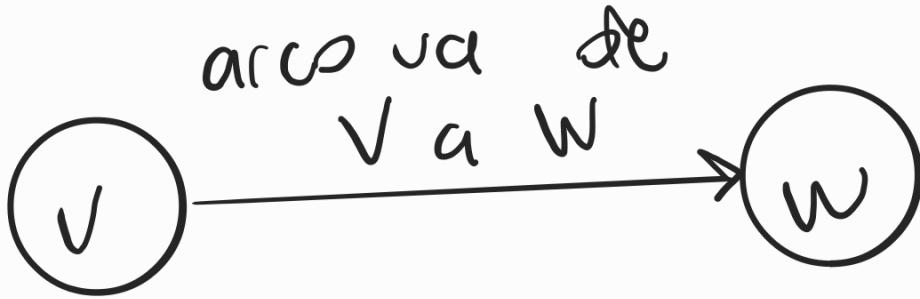
GRAFOS DIRIGIDOS

Un conjunto finito de vértices V
y de un conjunto de arcos A
 $G = (V, A)$

En un grafo dirigido, la arista es un par ordenado de vértices

ADYACENTES: Dos vértices tienen una arista que los conecta

CONECTADO: Cuando en un grafo existe un camino entre cualquier par de vértices



el nodo v
es la cola

el nodo w es
la cabeza

w es adyacente de v

CAMINOS

Un camino en un grafo dirigido
es una secuencia de vértices
 v_1, v_2, \dots, v_n tal que $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$ son arcos

La **longitud** del camino es el
número de arcos del camino.

Un **camino simple** si todos sus vértices, excepto tal vez el primero y el último, son **DISTINTOS**

Un **CICLO** es un camino simple de por lo menos largo dos, que empieza y termina en el mismo vértice

REPRESENTACIÓN

MATRIZ DE ADYACENCIAS:

Requiere un espacio mínimo del orden n^2 , siendo n la cantidad de vértices

Ventajas:

- Acceso rápido
- Sencillez
- Eficiente para grafos densos

Desventajas

- Ineficiencia de espacio
- Costoso en tiempo
- No es eficiente para añadir o elim. V

LISTA DE ADYACENCIAS

Requiere una cantidad de espacios proporcional a la suma de la cantidad de arcos mas la cantidad de vértices.

Ventajas

- Eficiencia en espacios
- Facil inserción y eliminación
- Util para algoritmos (BFS, Dijkstra, etc)

Desventajas

- Acceso mas lento
- Mas compleja
- Menos eficiente para grafos densos

MATRIZ

LISTA

Memoria

 $O(h^2)$ $O(h+a)$ Comprobar
existencia
arcos $O(1)$ $O(\text{grado}(n))$ Listas
Vecinos $O(h)$ $O(\text{grado}(n))$ Insertar

Eliminar

Costoso

Eficiente

Grafo
Disperso

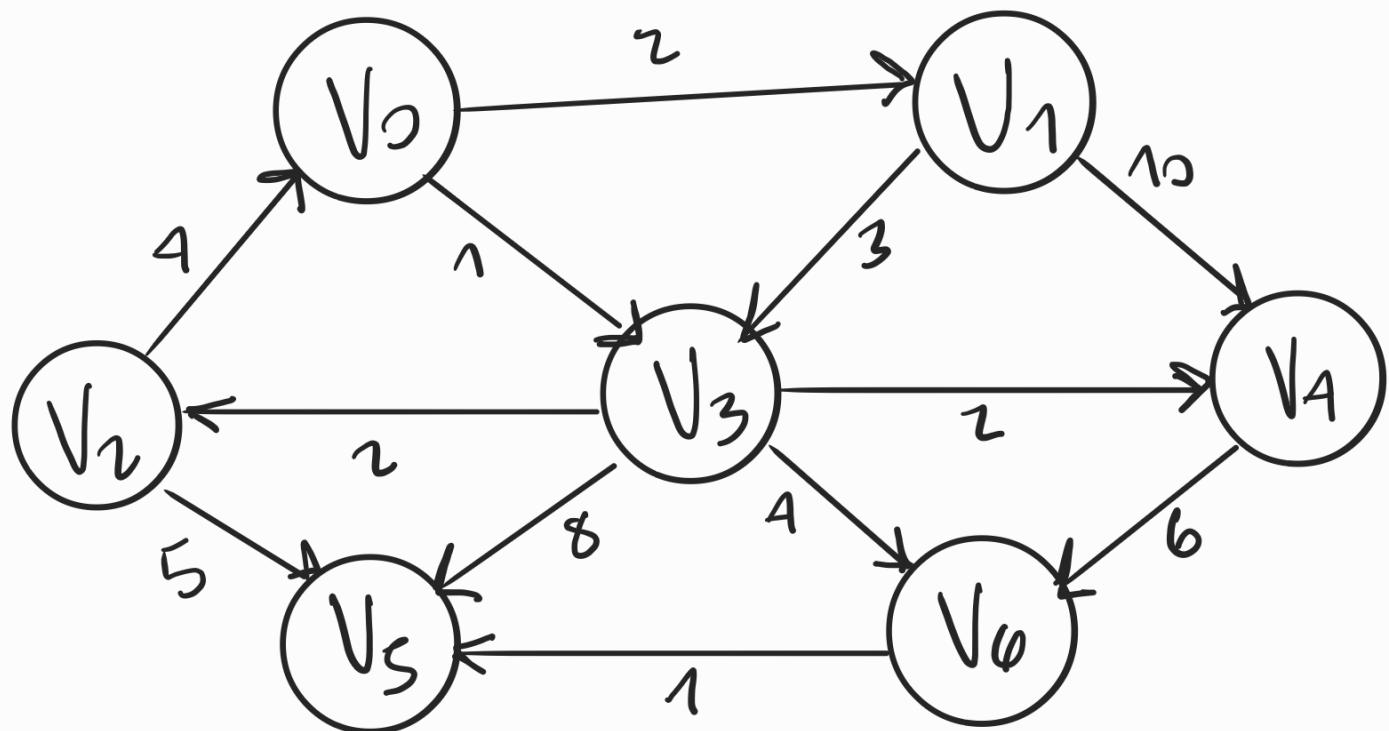
Ineficiente

Eficiente

Grafo
Denso

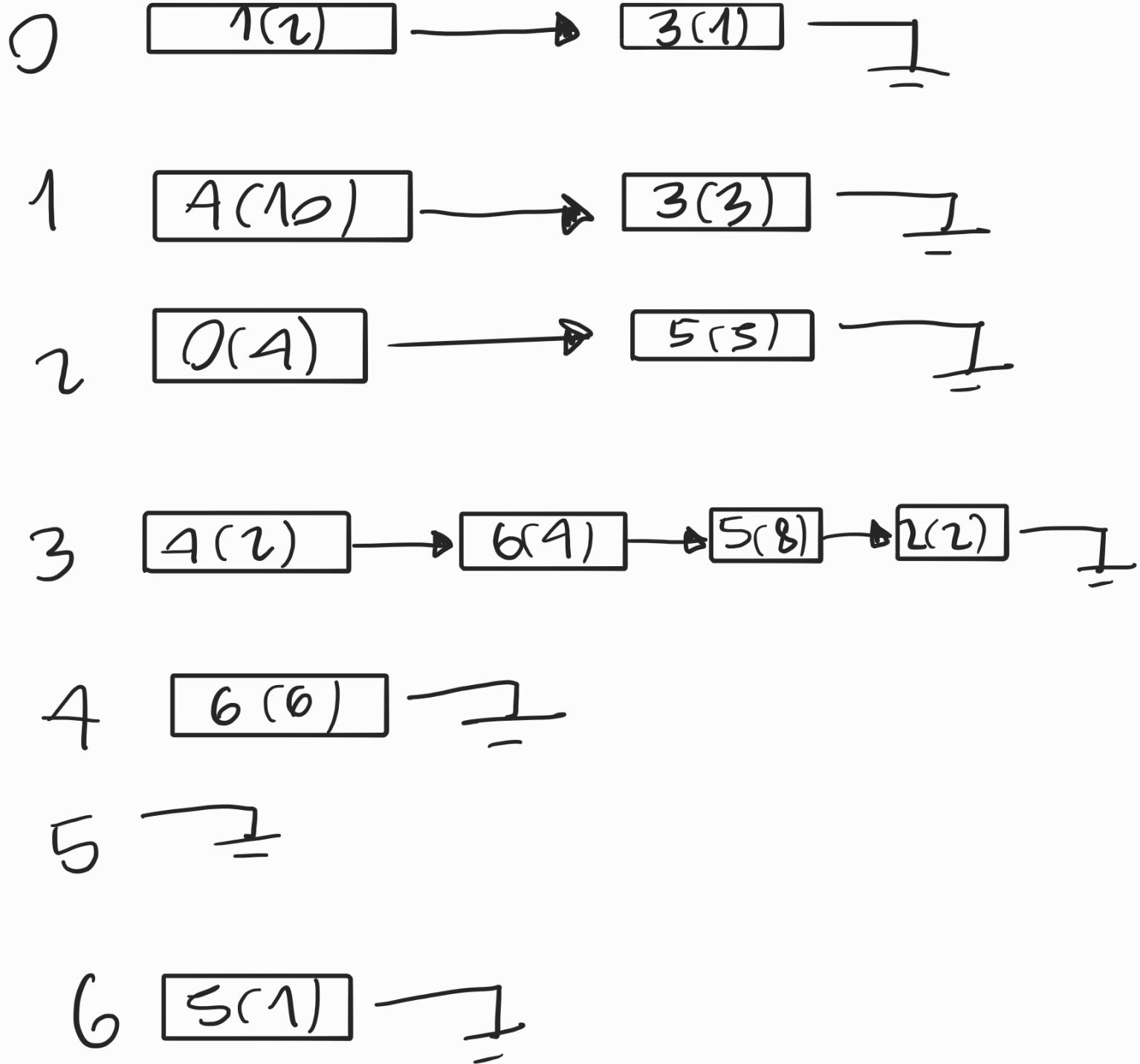
Eficiente

Ineficiente



0 1 2 3 4 5 6

	2		1			
0			3	10		
1					5	
2	4					
3		2		2	8	4
4						6
5						
6					1	



DJIKSTRA

- Problema: Determinar el costo del camino mas corto desde el origen hasta cada uno de los demás vértices V
- La longitud de un camino es la suma de los costos de los arcos del camino
- Técnica "AVIDA"

PSEUDO CODIGO

Fucion Dijkstra

COM

Inicializar S, D

$S = \{1\}$

para $i=2$ an hacer $D[i] = G[1,i]$

Mientras $V < S$ hacer

Elegir w perteneciente a $V-S$,

tal que distancia $D(w)$ sea un
mínimo

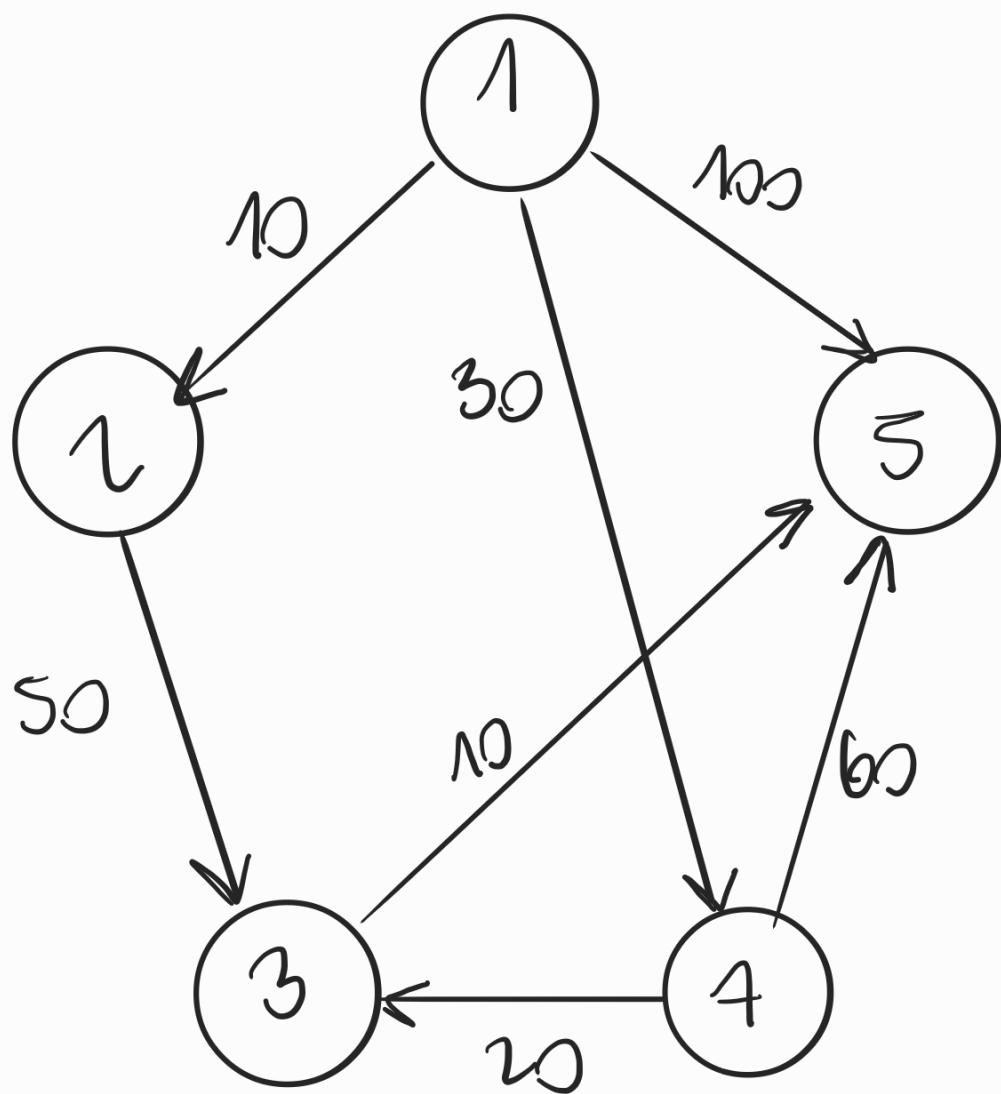
Agregar w a S

Para cada v perteneciente a
 $V-S$ hacer

$$D[v] = \min(D[v], D[w] + \text{costo}(w, v))$$

F_{IN} Mientras

F_{IN} {Dijkstra}



s	w	$D[2]$	$D[3]$	$D[4]$	$D[5]$
1	-	10	80	30	100
1,2	2	10	60	30	100
1,2,4	4	10	50	30	90
1,2,3,5	3	10	50	30	60
1,2,3,5	5	10	50	30	60

DIJKSTRA: RECUPERACIÓN DE CAMINOS

Función DIJKSTRA con caminos
COM

Iniciarizar S, D, P $S = \{1\}$;
para $i=2$ a n hacer $D[i] = c[1, i]$

Mientras $V < S$ hacer

Elegir w perteneciente a $V-S$,
tal que la distancia $D(w)$ sea
un mínimo

Agregar w a S

Para cada v perteneciente a
 $V-S$ hacer

Si $D[w] + \text{costo}(w, v) < D[v]$
entonces

$$D[v] = D[w] + \text{costo}(w, v) \quad y$$

$$P[V] = w$$

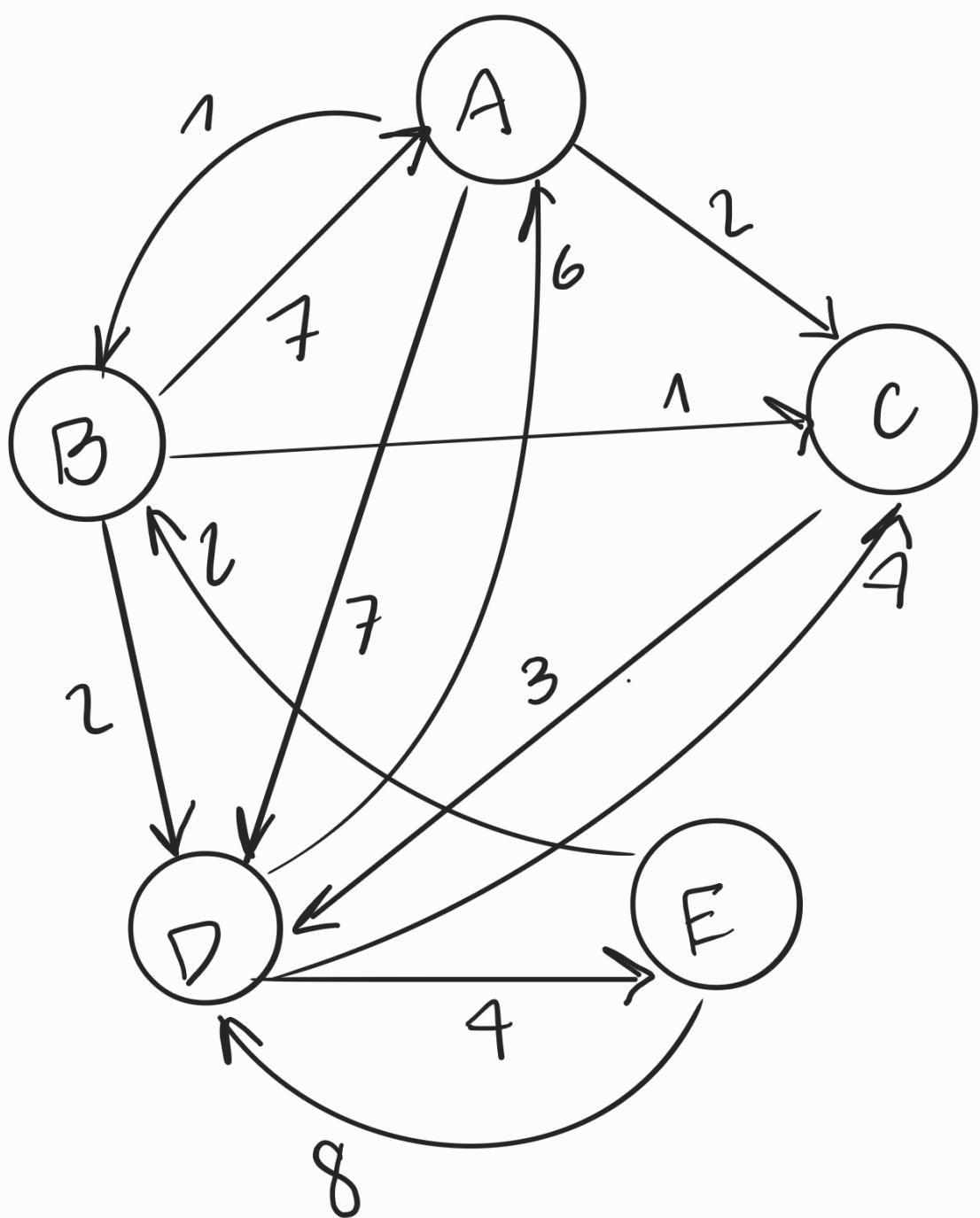
Fin Si

Fin Para Cada

Fin Mientras

Fin {Dijkstra}

	A	B	C	D	E
A	-	1	2	7	-
B	7	-	1	2	-
C	-	-	-	3	-
D	6	-	4	-	4
E	-	2	-	8	-



S	W	D[B]	D[C]	D[D]	D[E]
A	-	1	2	7	8
A,B	B	1	2	3	8
A,B,C	C	1	2	3	8
A,B,C,D	D	1	2	3	7
A,B,C,D,E	E	1	2	3	7

FLOYD

Problema: Obtener una tabla que indique el menor camino entre todos los pares de los vértices.

- Se usa una matriz de $n \times n$ en la que se calcular las longitudes de los caminos más cortos.

Se compara si el camino directo entre dos vértices es menos costoso que pasando por otro vértice

PSEUDO FLOYD

Funcion Floyd (A: Array[1..n, 1..n] of
real ;
C: array [1..n, 1..n] of
real;)

Var i, j, k: integer;
begin
for i:=1 to n do
 for j:=1 to n do
 A[i,j]:=c[i,j]; P[i,j]:=0;
 for i:=1 to n do A[i,j]:=0;
 for k:=1 to n do
 for i:=1 to n do
 for j:=1 to n do
 if (A[i,k]+A[k,j]) < A[i,j]
 then A[i,j]:=A[i,k]+A[k,j];
 P[i,j]:=k;
end;

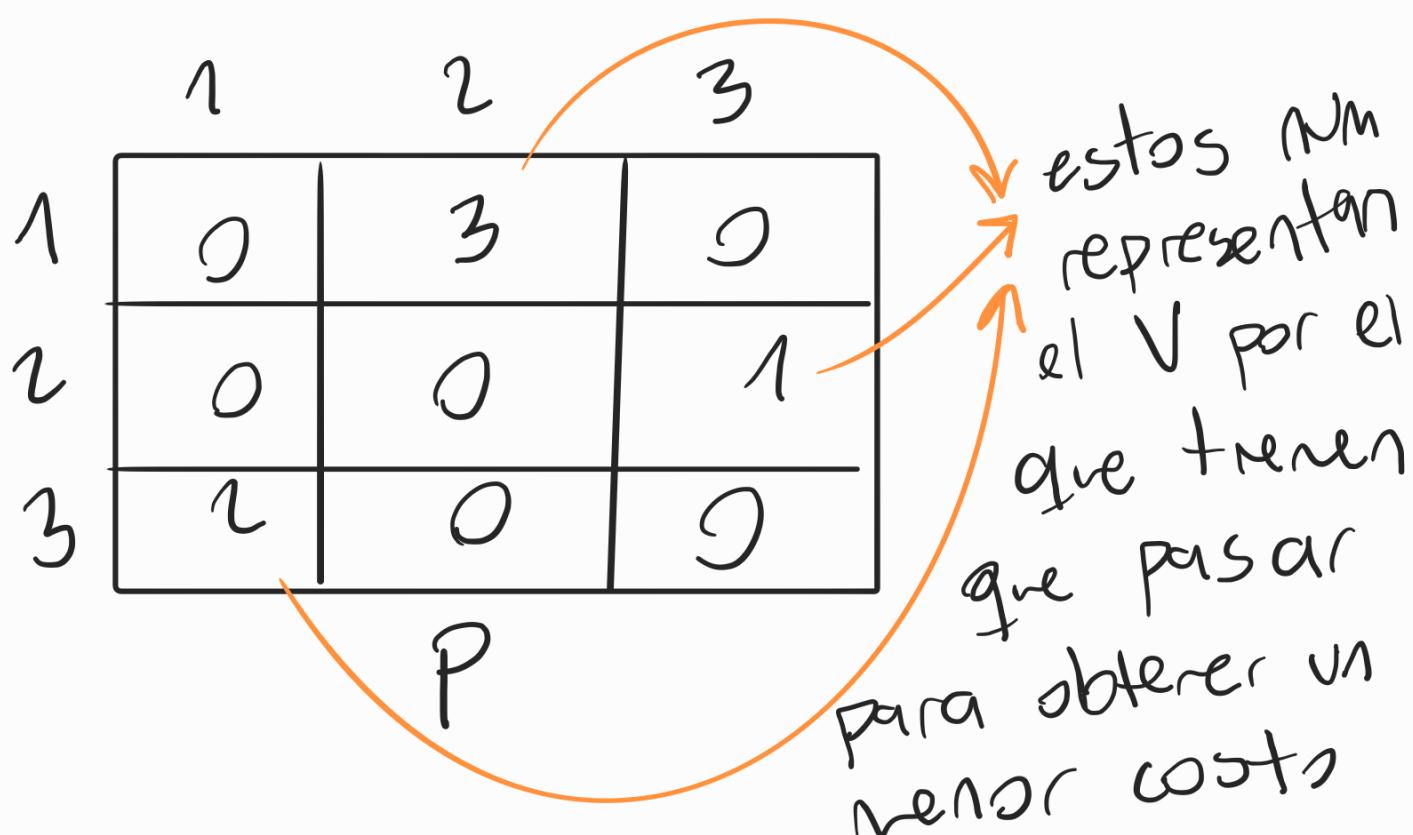
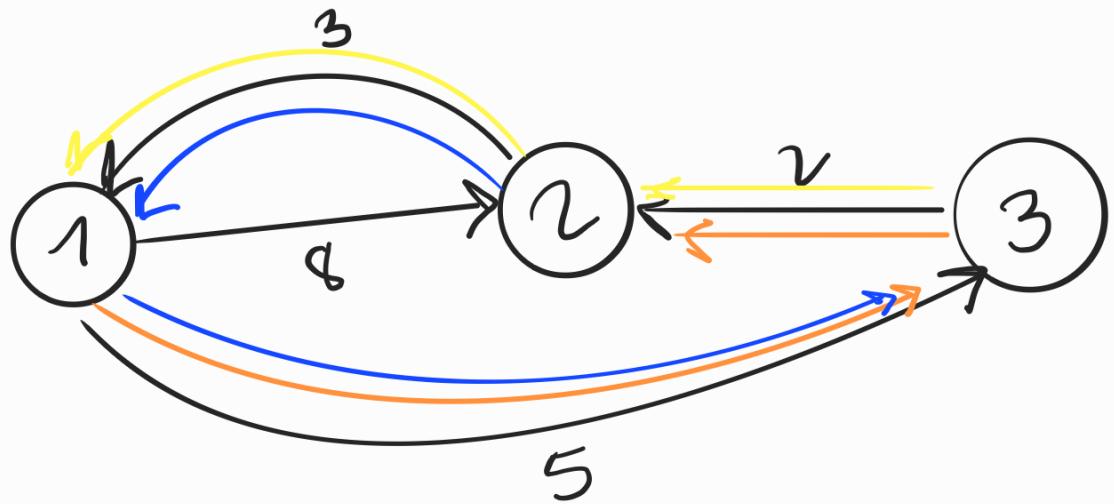
Complejidad

Tiempo: $O(n^3)$, porque se hace n
tres bucles uno dentro del otro

Espacio: $O(n^2)$ para almacenar la
matriz A.

RECUPERACIÓN DE CAMINOS

Agregamos una matriz P en
donde $P[i,j]$ contiene aquél vértice
k que determinó que Floyd encontrara
el mismo valor para $A[i,j]$



Procedure camino (i,j:integer);
var k: integer
begin k := P[i,j];
if k=0 then Salir;
camino (i,k);
imprimir(k);
camino (k,j);
end; {camino}

EXCENTRICIDAD

Se define como la máxima de todas las longitudes mínimas de los caminos entre cada uno de los otros nodos y el nodo V.

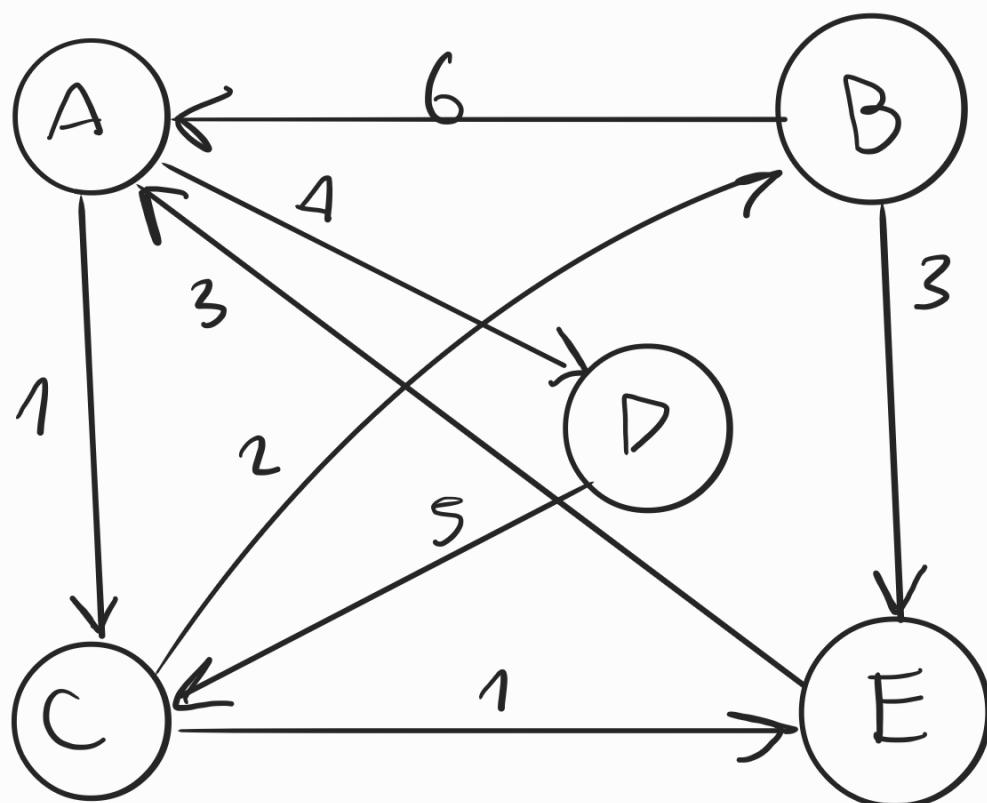
- Indica que tan lejano está ese nodo respecto al nodo más distante del grafo.
- Si un nodo tiene excentricidad baja, significa que está cerca de todos los otros nodos.
- Si tiene excentricidad alta, al menos hay un nodo muy lejano a él.

• El **CENTRO** de G es un vértice de mínima excentricidad

Para obtener el centro de grafo

- Aplicar Floyd

- Encontrar el máximo valor de cada columna
- Encontrar el V con excentricidad mínima



	A	B	C	D	E
A	0	∞	1	4	∞
B	6	0	∞	∞	3
C	∞	2	0	∞	1
D	∞	∞	5	0	∞
E	3	∞	∞	∞	0

Aplicamos Floyd: →

	A	B	C	D	E
A	0	3	1	4	2
B	6	0	7	10	3
C	8	2	0	8	1
D	13	7	5	0	6
E	3	6	4	7	0

Max: 13 | 7 | 7 | 10 | 6

El centro del grafo es E

CERRADURA TRANSITNA

ALGORITMO de WARSHALL

- Para saber si existe un camino que vaya del V_i al V_j
- La matriz de costos indicará 1 si hay arco, 0 si no lo hay
- Completa la matriz con todos los caminos indirectos posibles.

Procedure Warshall (A: array [1..n, 1..n] of boolean;
(C: array [1..n, 1..n] of boolean;)

i, j, k: integer;

COM

for i := 1 to n do

for j := 1 to n do

A[i, j] := G[i, j];

for k := 1 to n do

for i := 1 to n do

for j := 1 to n do

if A[i, j] = false

then A[i, j] := A[i, k] and A[k, j];

FIN

BUSQUEDA EN PROF.

- Se selecciona un vértice V como vértice de partida y se marca como visitado
- Luego se recorre cada vértice no visitado adyacente a V recursivamente.
- La búsqueda se completa cuando se visitan todos los vértices adyacentes. Si quedan vértices sin visitar, se selecciona otro vértice de partida y se repite el proceso

método Tvertice , bpf():

w: TVertice;

COM

(1) Visitar();

Para cada vecino w hacer

(2) Si no (w.visitado()) entonces

(3) w.bpf

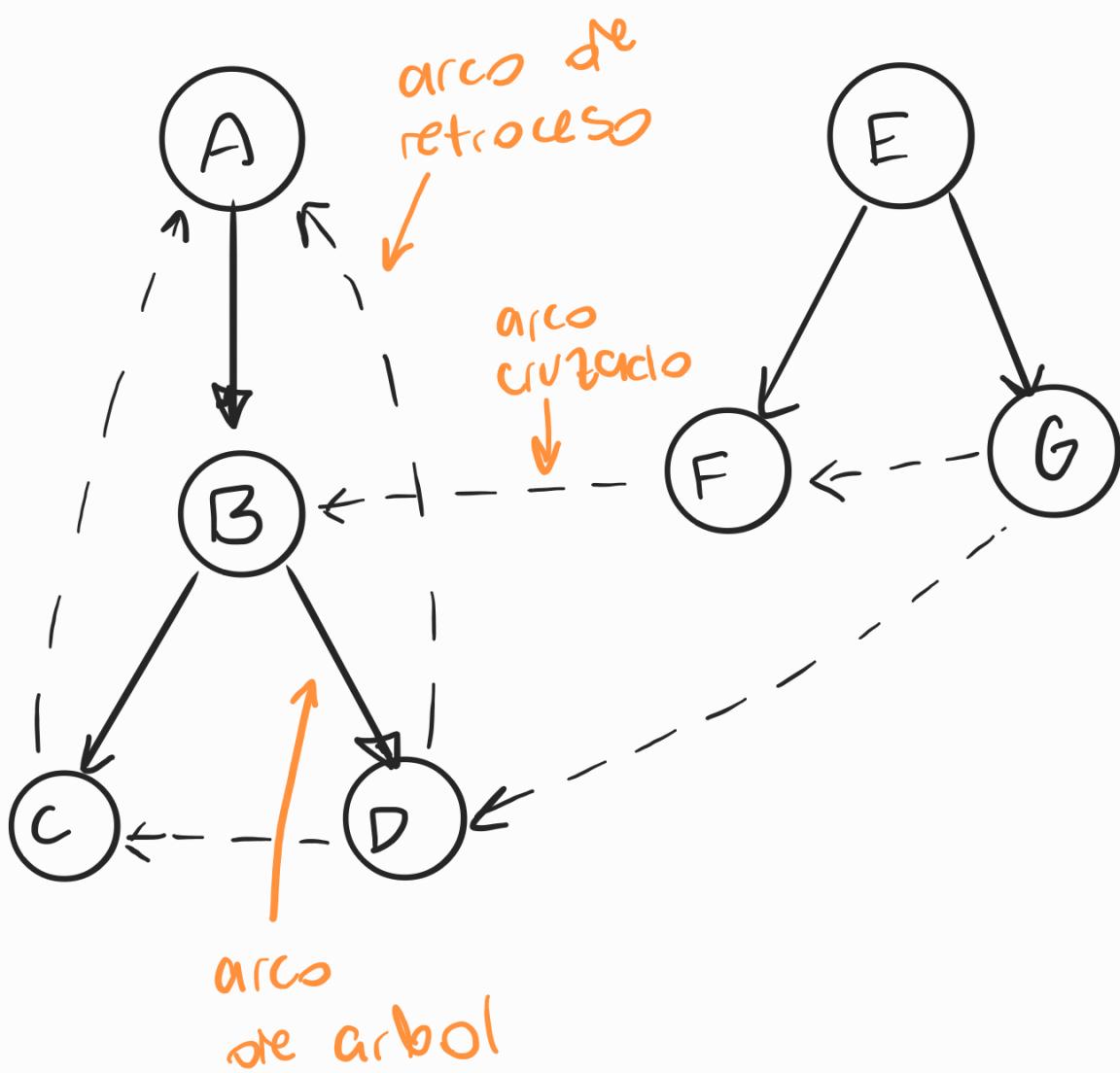
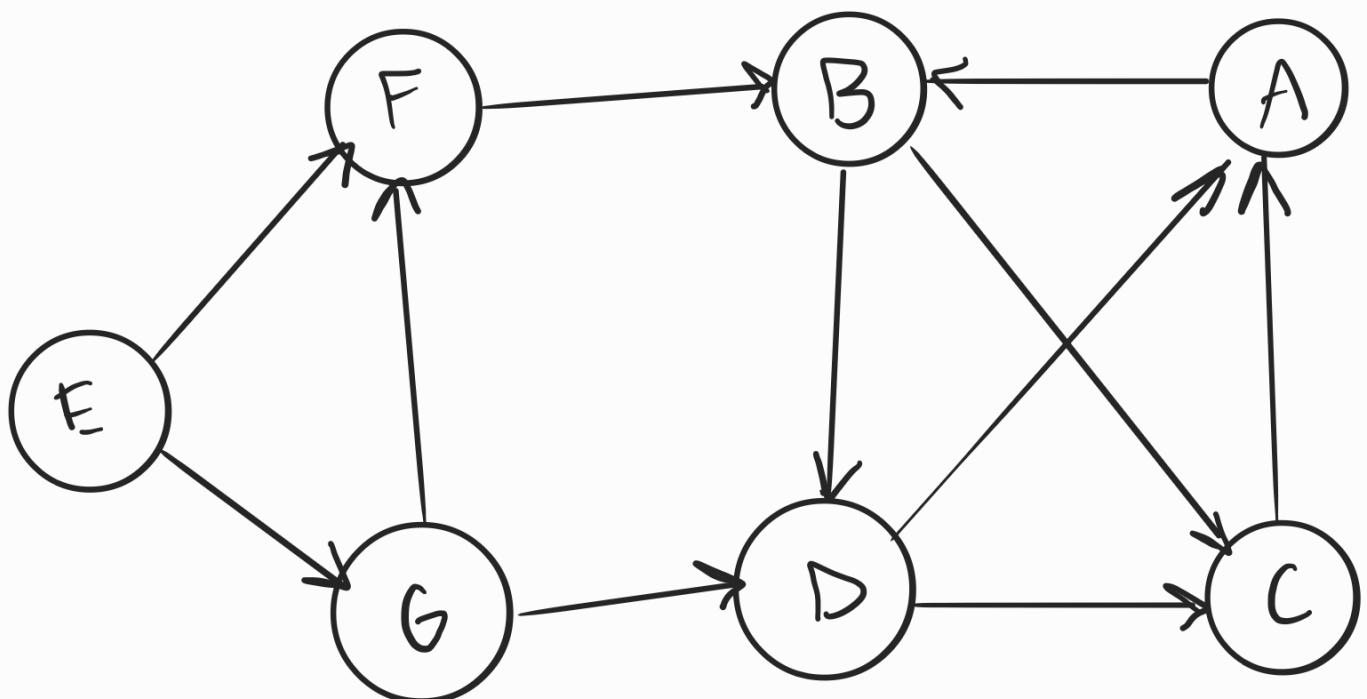
FIN SI

Fn para cada

FIN {bpf}

Orden de tiempo de ejecución:

O(n), ya que es necesario
recorrer cada arco



Arco de arbol: Va a un vertice nuevo

" " retroceso: Va a un ancestro ya visitado

" " avance: Va a uno de sus descendientes ya visitados

Arco cruzado: Va a un vertice que no es ni ancestro ni descendiente.

metodo Camino (Destino : Vertice ;)
ElCamino : T(Camino))

w : Vertice

COM

Visitar(); Agregar("this", ElCamino)

Para Cada adyacente w

Si w = Destino entonces

Guardar (ElCamino + Destino)

SINO

Si no visitado(w) entonces

Camino(w, Destino, ElCamino)

FIN SI

FIN SI

FIN Para Cada

Quitar ("this", ElCamino)

FIN

GRAFOS DIRIGIDOS ACICLICOS

- Es un grafo dirigido SIN CICLOS
- Prueba de Aciclicidad:
Se realiza una BPF y si se encuentra un arbol de retroceso, entonces tiene un ciclo

CLASIFICACION TOPOLOGICA

Es una manera de ordenar linealmente los vertices de un grafo aciclicos basado en las dependencias entre uno y otro.

procedure ClasificacionTopologica();

w:Vertice

COM

(1) visitar();

(2) Para cada adyacente w hacer

Si no (w.visitado()) entonces

w.ClasificacionTopologica()

FIN SI

Fin Para Cada

Imprimir();

FIN;

