DFS($G = (V, E)$)

$\forall v \in V$ color(v) $\leftarrow$ white

$\pi(v) \leftarrow$ nil

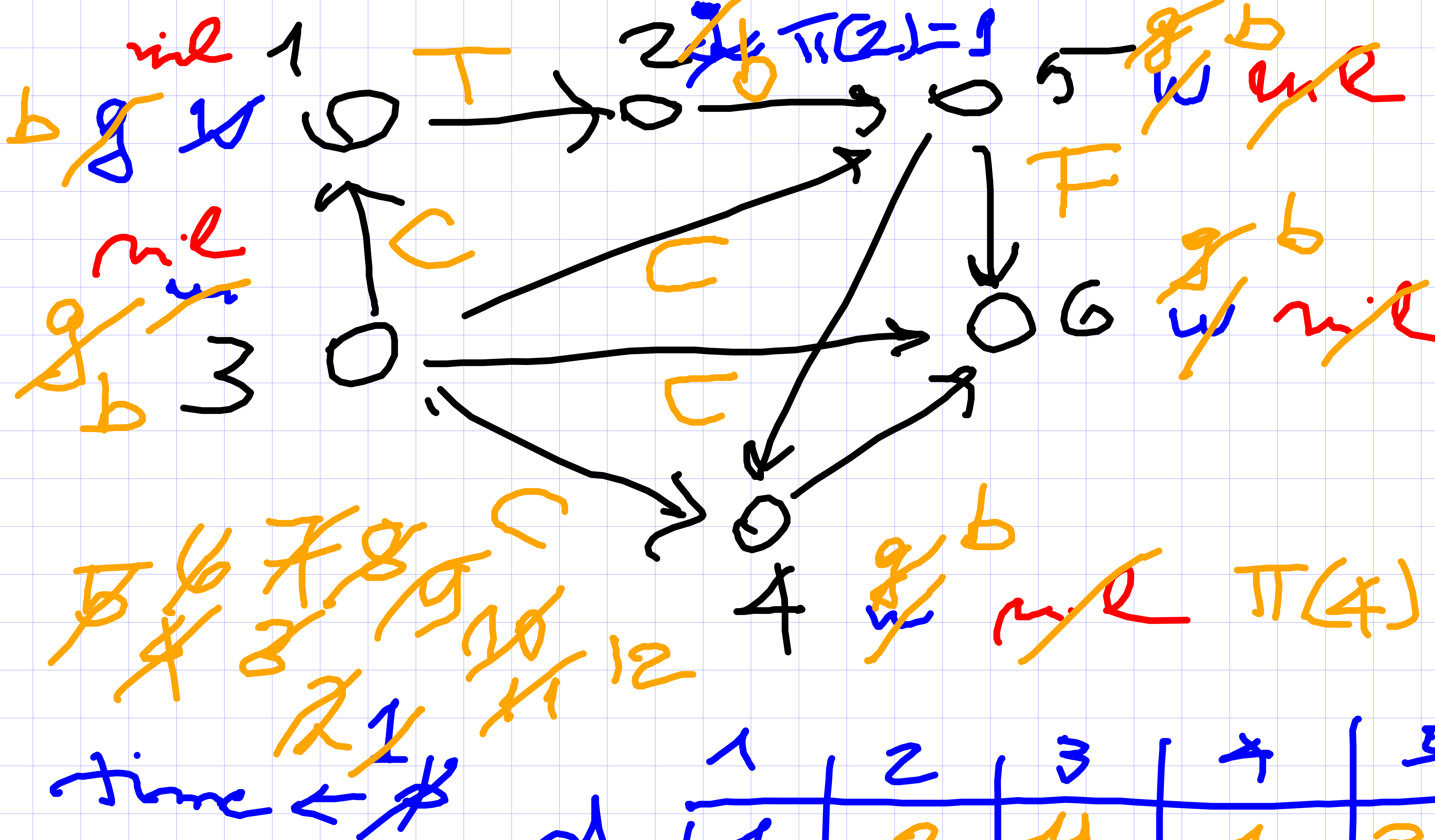
time $\leftarrow 0$

for $i \leftarrow 1$ to $|V|$ do

if color(i) = white then

DFS-visit(G, i, time)

```
DFS-visit (G, v, time);
    color(v) ← grey
    time++;
    d(v) ← time;
    ∀ u ∈ Adj(v) do
        if color(u) = white then
            (v, u) ∈ T
            π(u) ← v
            DFS-visit (G, u, time)
        if color(u) = gray then
            (v, u) ∈ Backword
        if color(u) = black and
            d(v) < d(u)
            then (v, u) ∈ forword
            else (v, u) ∈ cross
    color(v) ← black; time++; f(v) ← time
```



$\pi(2)=1$

$\pi(5)=2$

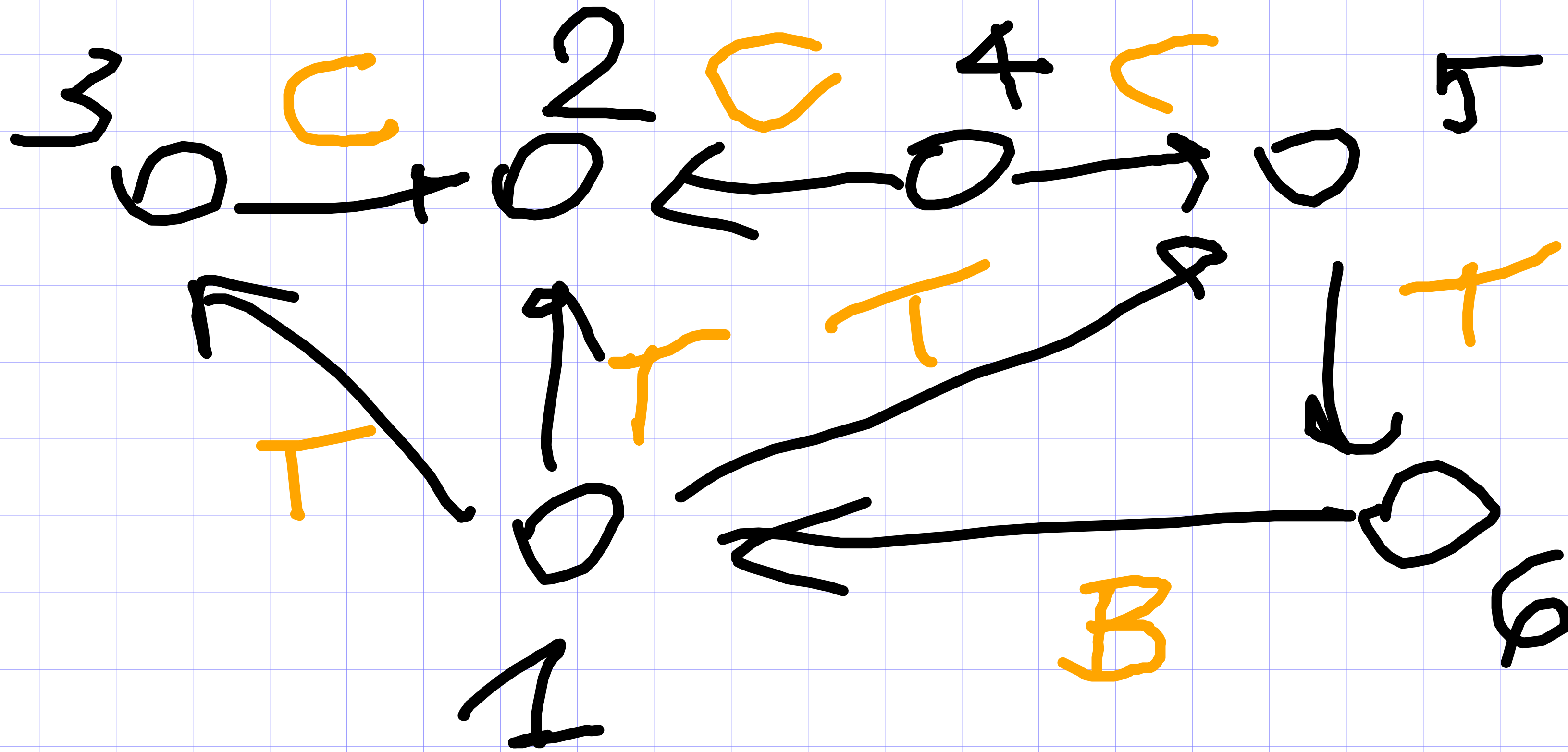
$\pi(6)=4$

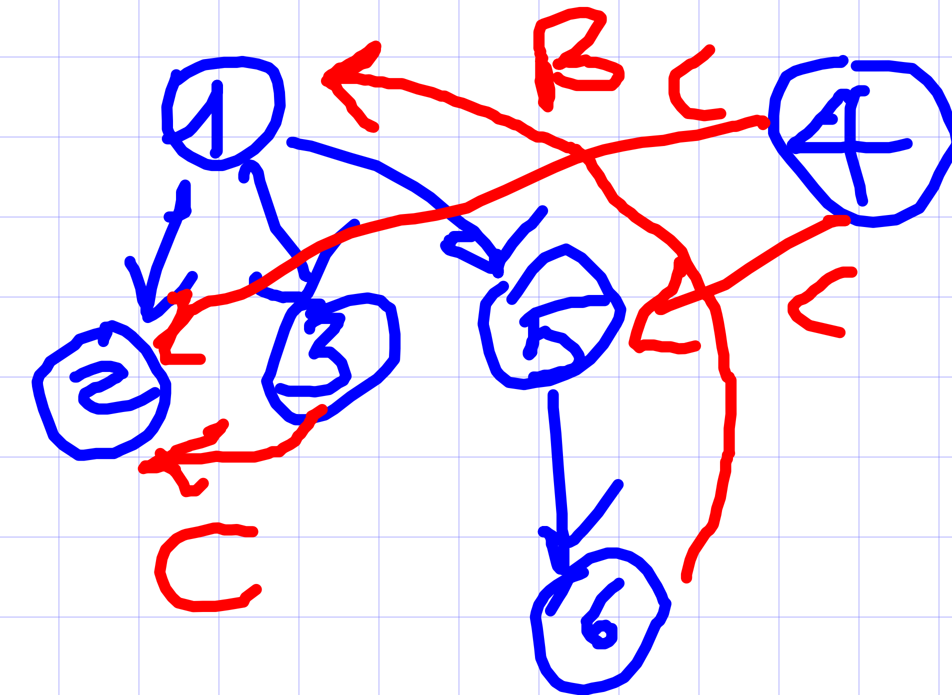
$\pi(4)=5$

time → ~~8~~

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

	1	2	3	4	5	6
d	1	2	11	4	3	5
f	10	9	12	7	8	6

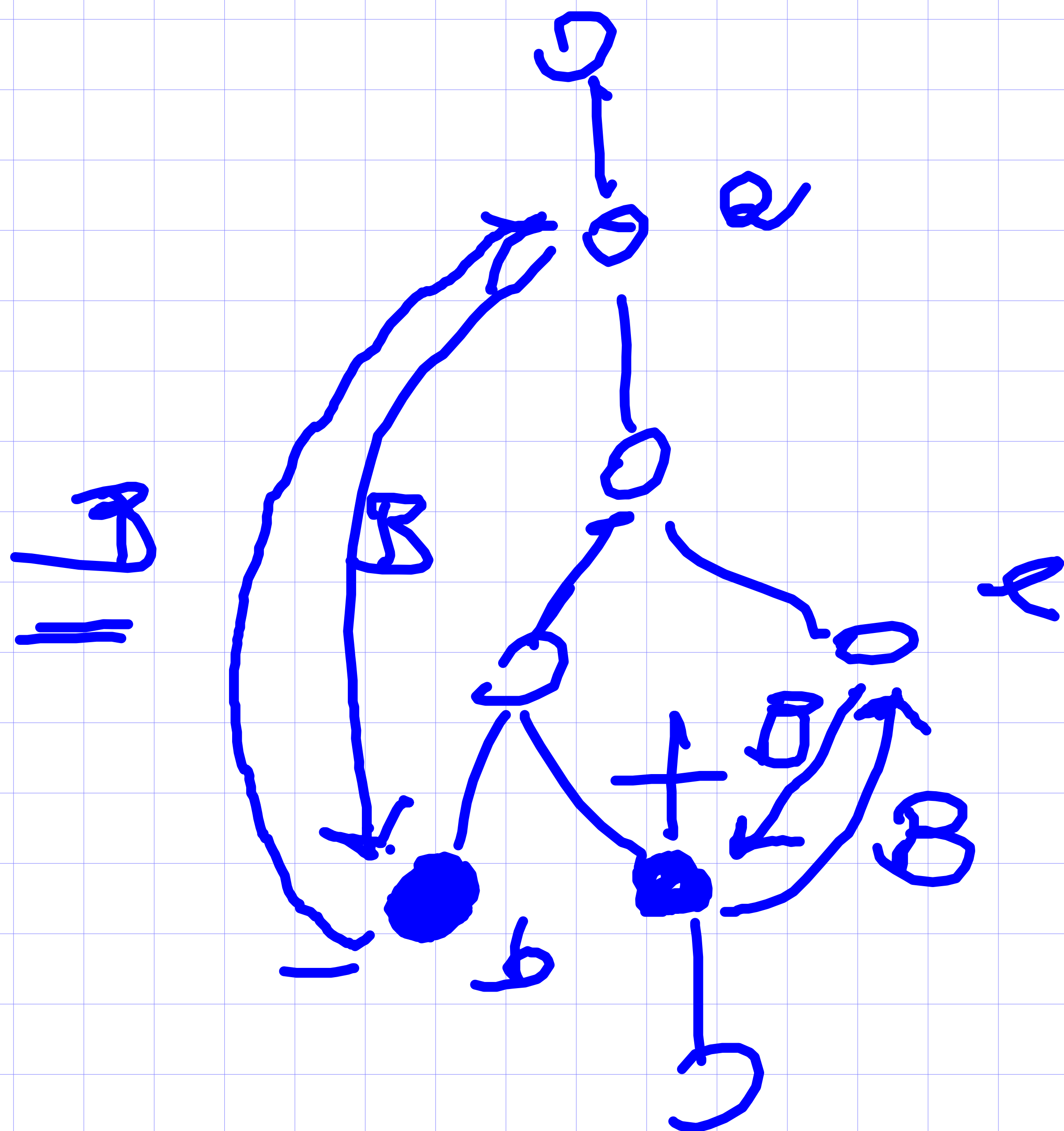




	1	2	3	4	5	6
color	b	b	b	~~g~~ b	b	b
π	∅	1	1	∅	4	5
d	1	2	4	11	6	7
f		3	5	12	9	8

time ← ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ 12

G non orientato



b è diventato nero
(b,a) già stato visitato
ed è arco in
B

G . DFS(G) ho solo archi in T e in B

$\Rightarrow$ G è non orientato

FALSO

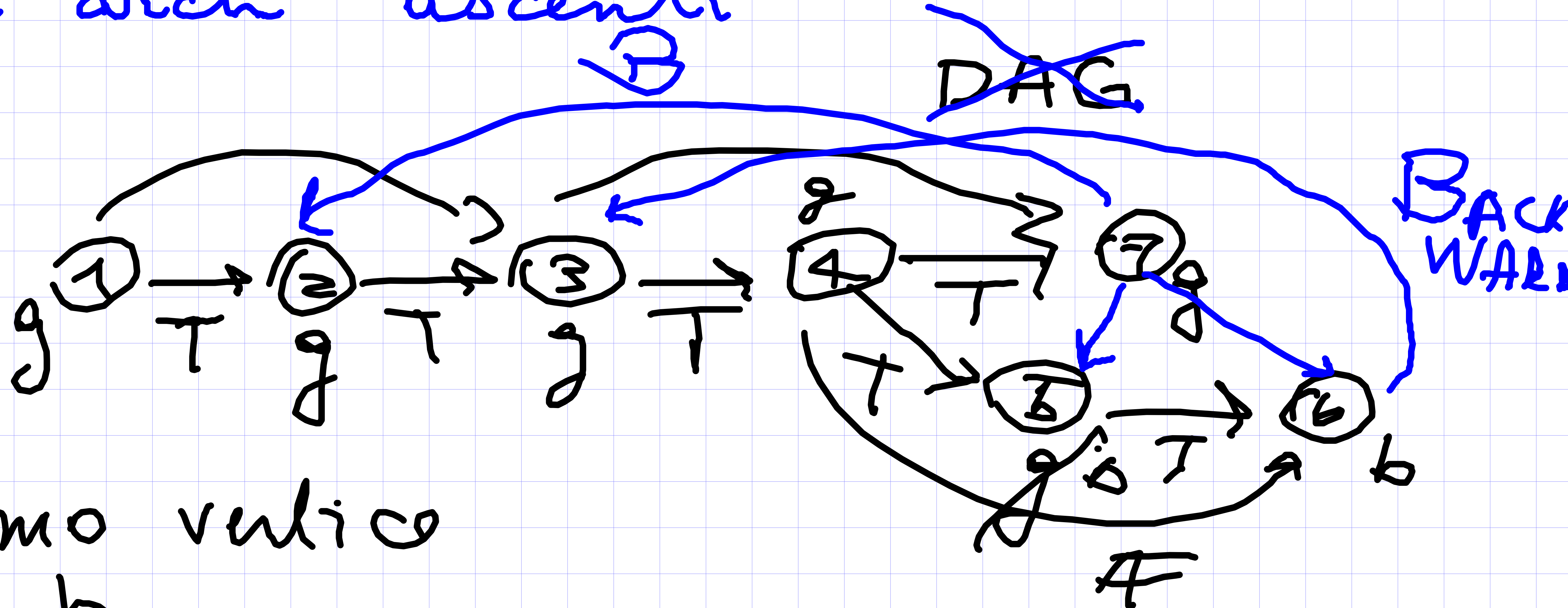
Direct $\boxed{\text{acyclic}}$ graph DAG

verificare l'$\exists$ di

Arch Backward

$\exists B$	$\nexists B$
NON È DAG	È DAG

Un <u>grafo orientato aciclico</u> ha almeno un <u>sink</u>, cioè un vertice senza archi uscenti



ultimo vertice

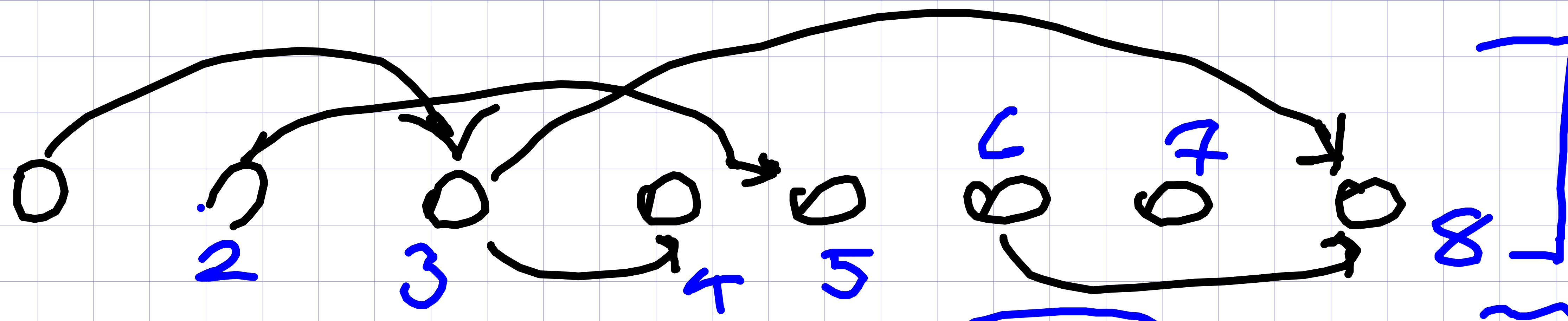
g, b

no sink → no DAG

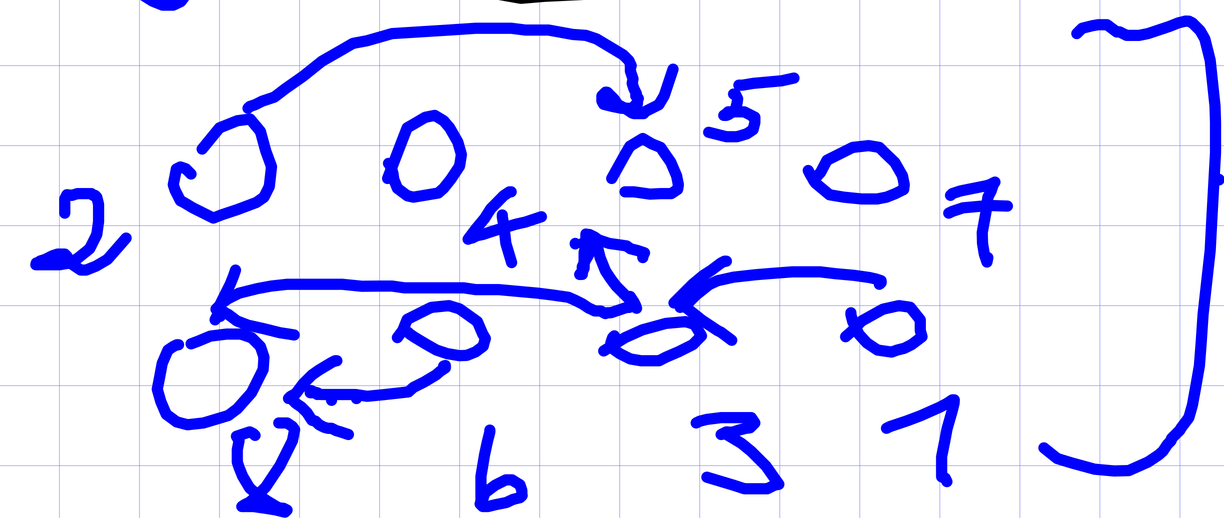
SORT TOPOLOGICO o linearizzazione

è un ordinamento dei vertici:

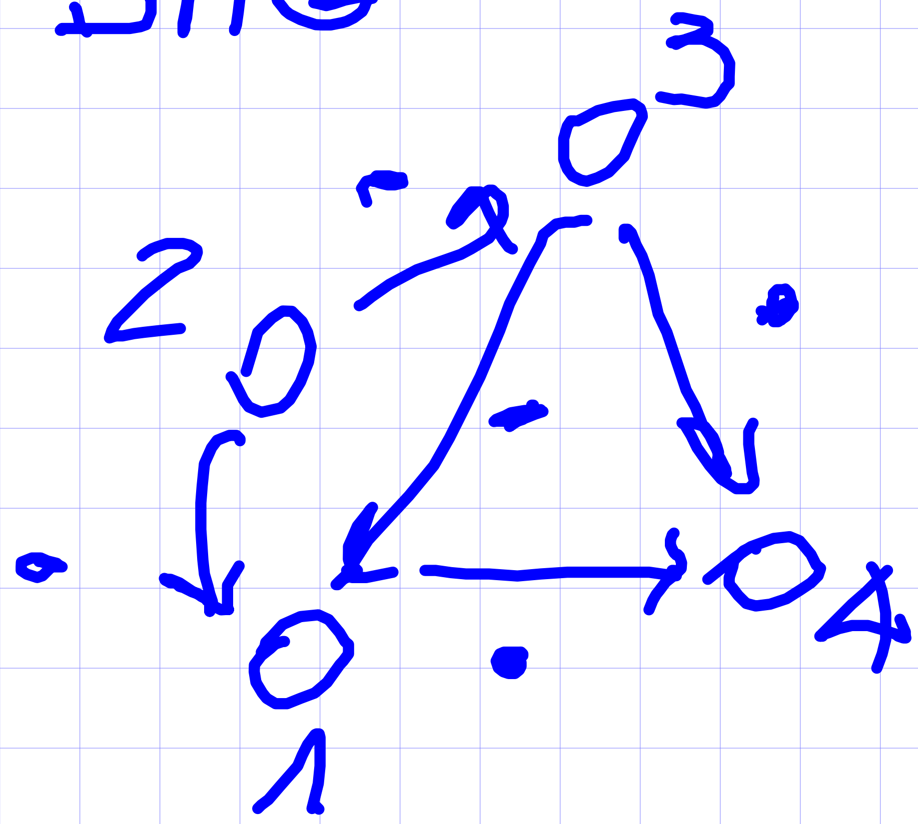
$\forall e = (u, v)$ u precede v nell'ordinamento dei vertici

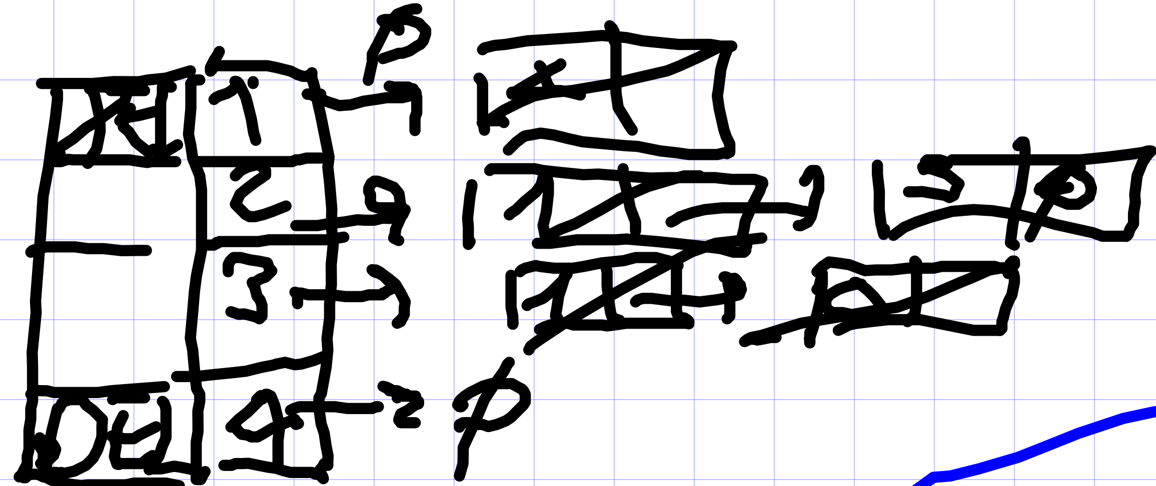


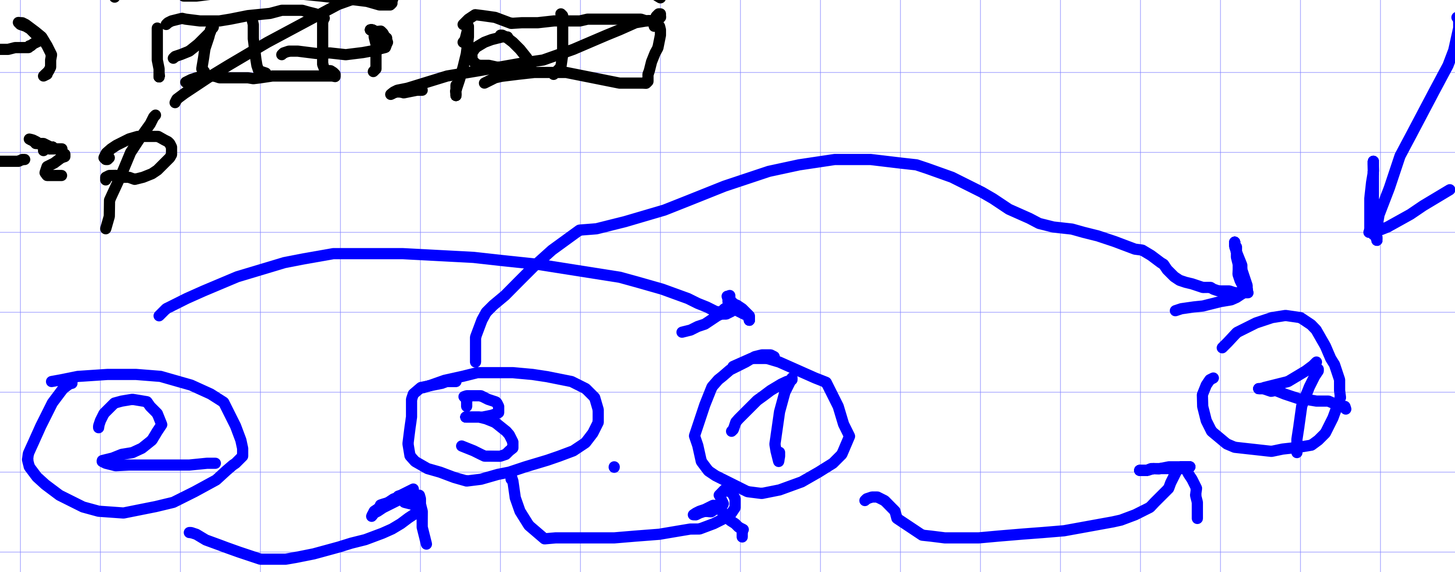
$e = (1,3)$, $(2,5)$



DAG



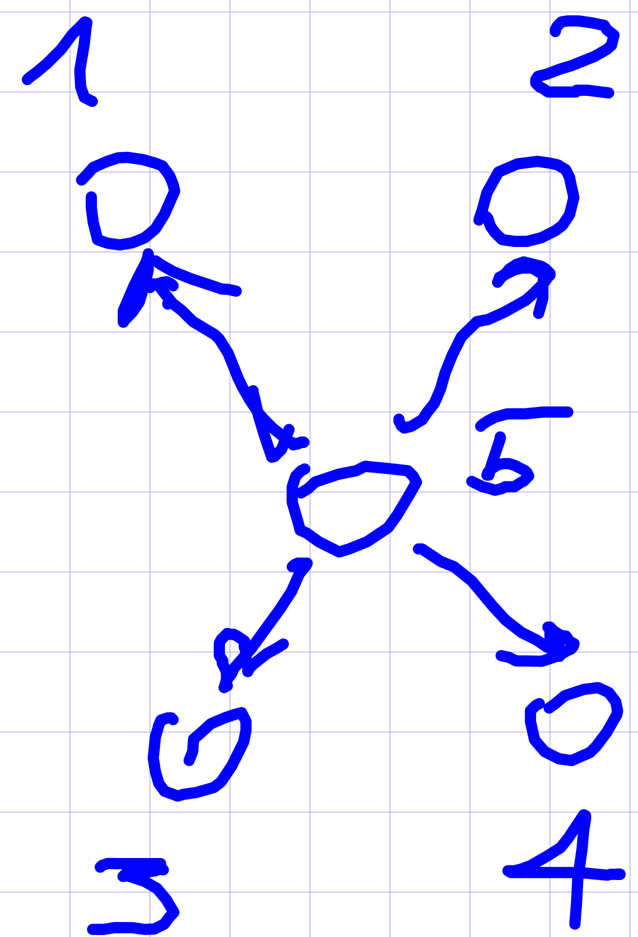




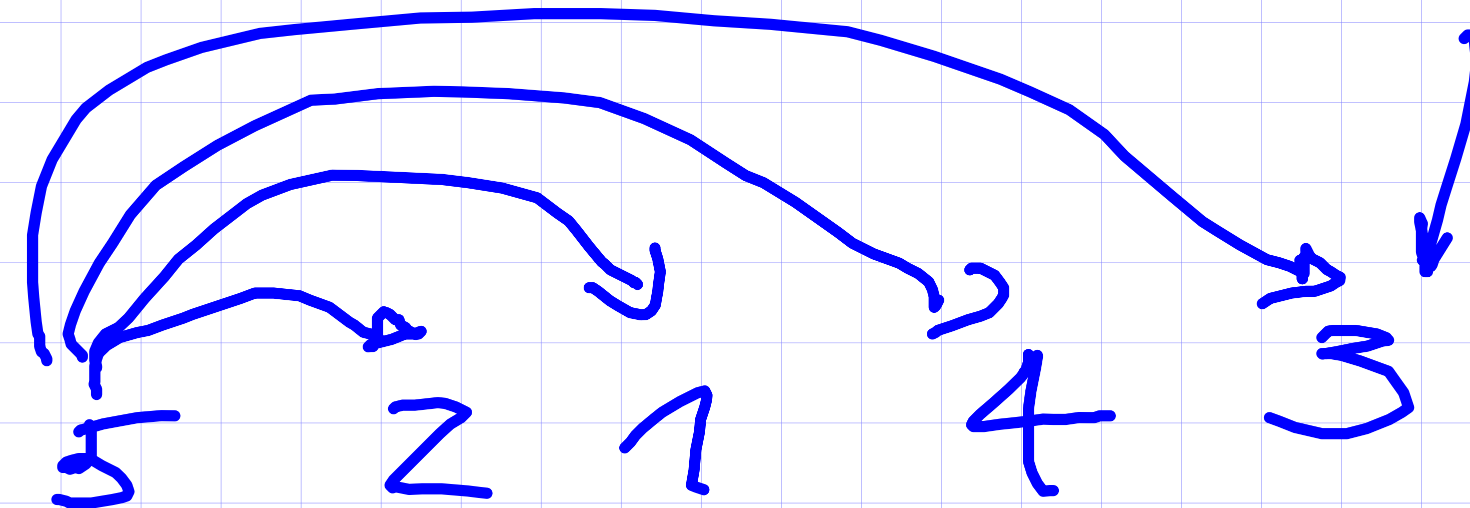
pos : 1 2 3 4

v	pos
2	1
1	3
3	2
4	4

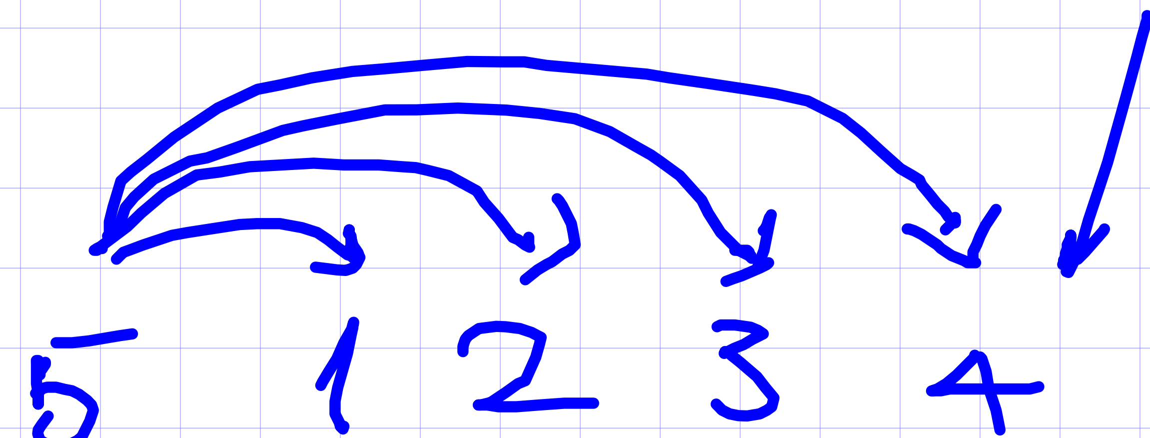
SORT TOPOLOGICO
TROVARE 'pos'



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



	201
5	1
2	2
3	5
4	4
1	3



$t \leftarrow |V|$ $\quad O(V^3)$ $\quad \leftarrow$ Lista adiacenza / Matrice adiacenza

REPEAT

let s be a sink in G $\quad O(|V|)$ $\quad O(V^2)$

pos(s) ← t ; t-- $\quad O(1)$ $\quad O(1)$

delete (s) da G $\quad O(V+E)$ $\quad O(V)$

// elimina in G s e gli archi
entranti in s //

until G = {∅}

$\Rightarrow O[V(V+E)]$

SORT TOPOLOGICO (G)

1. DFS (G) $\quad \forall v \in V : f(v)$

2. ordina i vertici in senso decrescente di fine visita

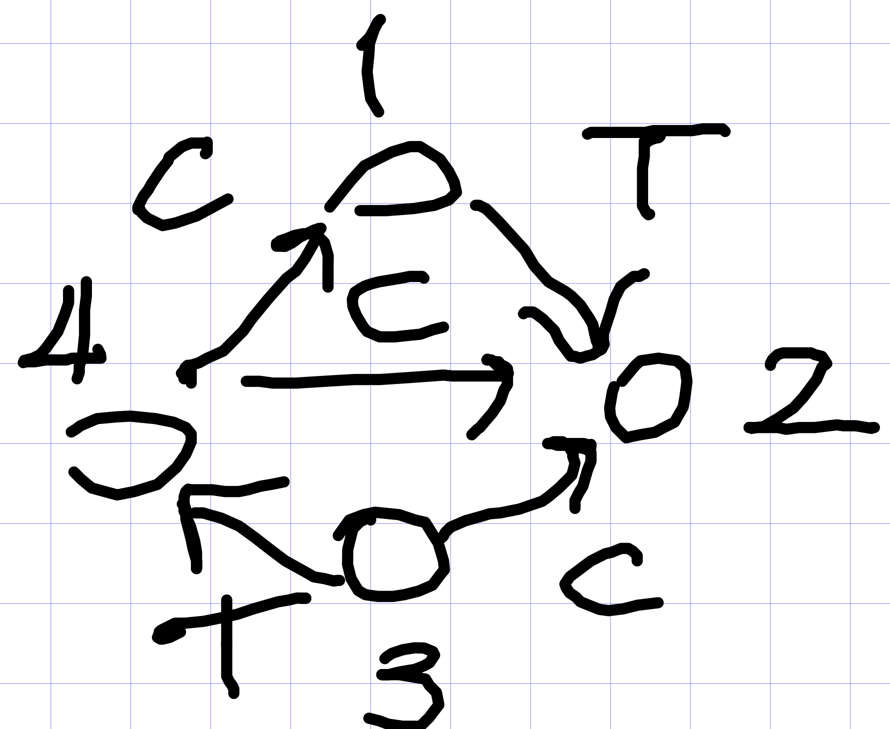
$2 \leq f(v) \leq 2|V|$

Complessità

COUNTING SORT

L.A. $\quad O(V+E) + O(V \quad)$

M.A $\quad O(V^2) + O(V)$

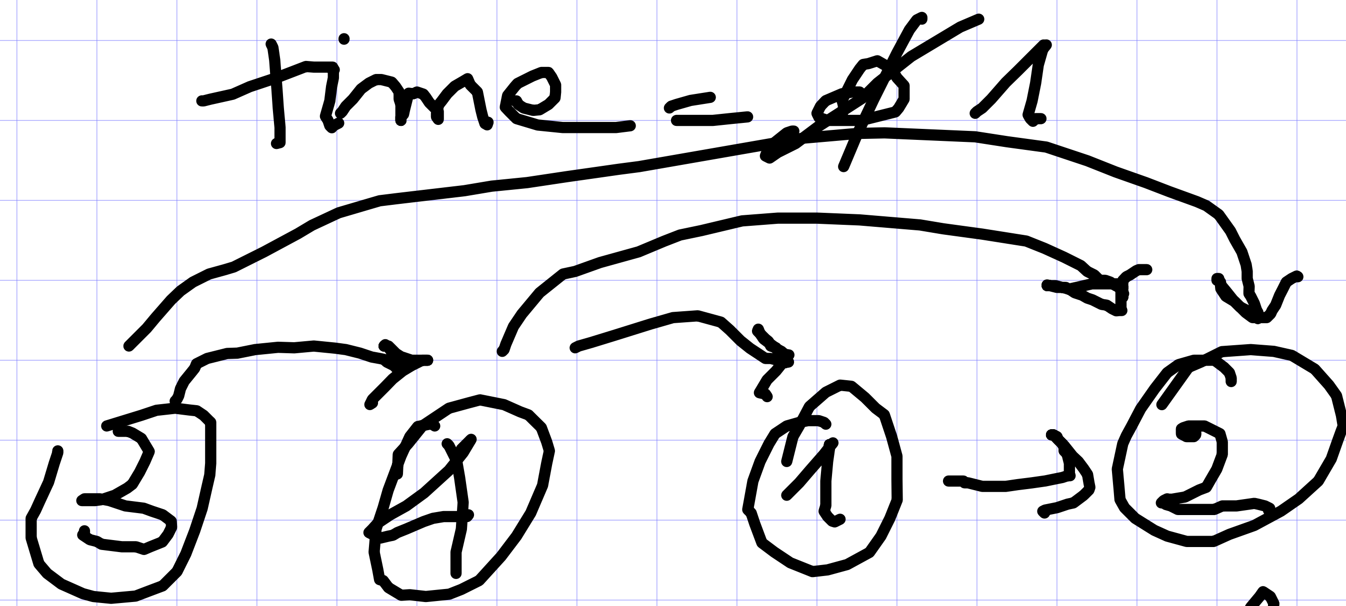


color	π	d	f
g	$\emptyset$	1	4
b	1	2	3
g	$\emptyset$	5	8
g	3	6	7

Rows are labelled 1, 2, 3, 4.

DFS (G)

time = $\cancel{0}$ 1



f = 8 f = 7 f = 4 f = 3

pos(3) = 1
pos(4) = 2
pos(1) = 3
pos(2) = 4

SORT TOPOLOGICO $(G : DAG)$

1. DFS(G) calcolando $\forall v \in V: f(v)$ fine visi

   + INSERISCI in testa ad una lista L ogni nodo che diventa nero

2. return L

L.A $O(V+E)$

M.A $O(V2)$

$L \to \boxed{3 \mid\ } \to \boxed{4 \mid\ } \to \boxed{1 \mid\ } \to \boxed{2 \mid\ }$

Cercare un sink in G rappresentato da
M.A.
$t \leftarrow 1$; trovato $\leftarrow$ false
while ($t \leq |V|$ and ($\neg$ trovato)
{ if $\neg \underbrace{\text{Sink-Test}(G, t)}_{O(|V|)}$ then $t{+}{+}$
else trovato $\leftarrow$ TRUE
}
return t

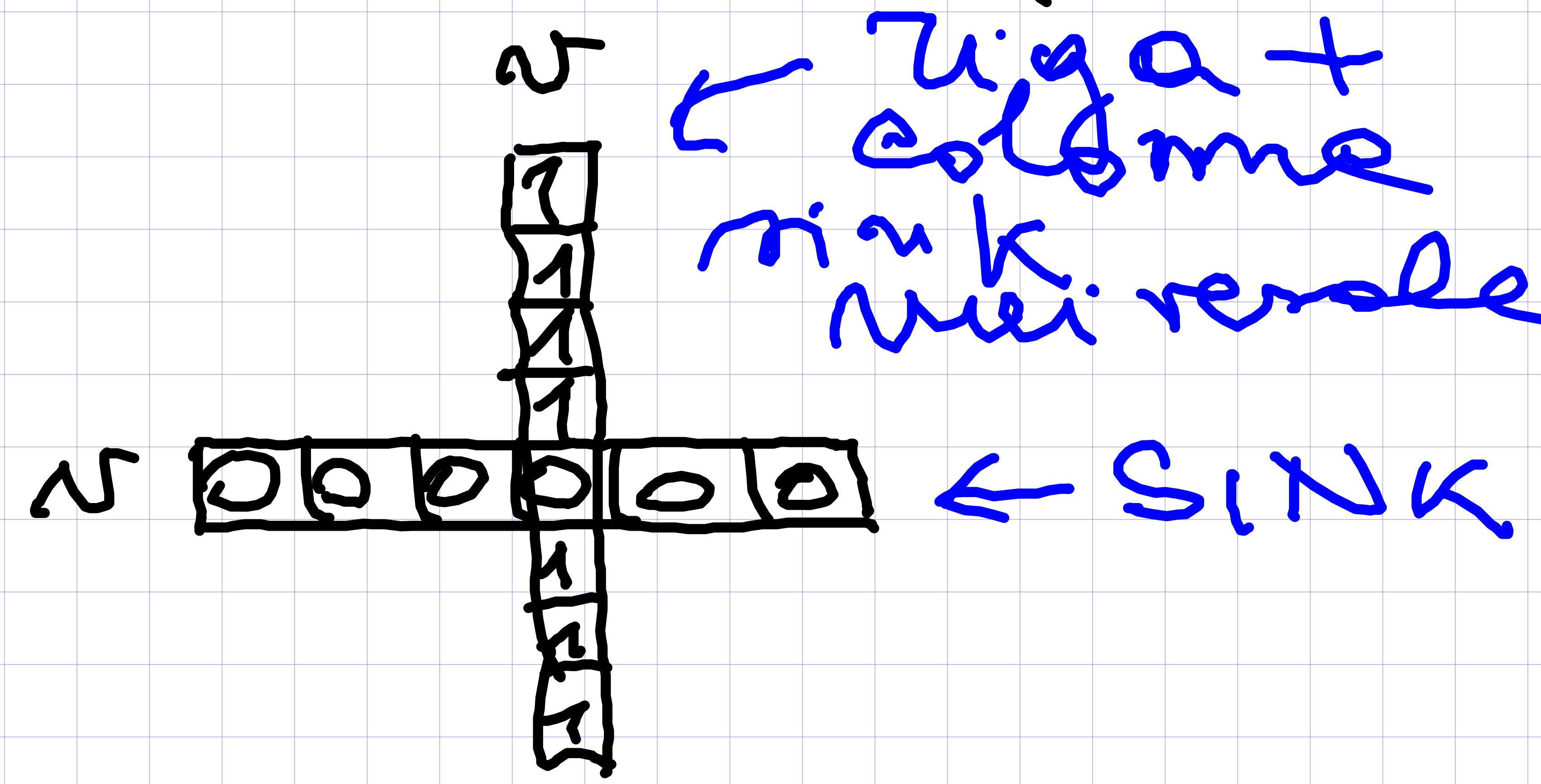
Complessità worst-case $O(V^2)$
Questo algoritmo trova un sink
(non cerca sink universale)

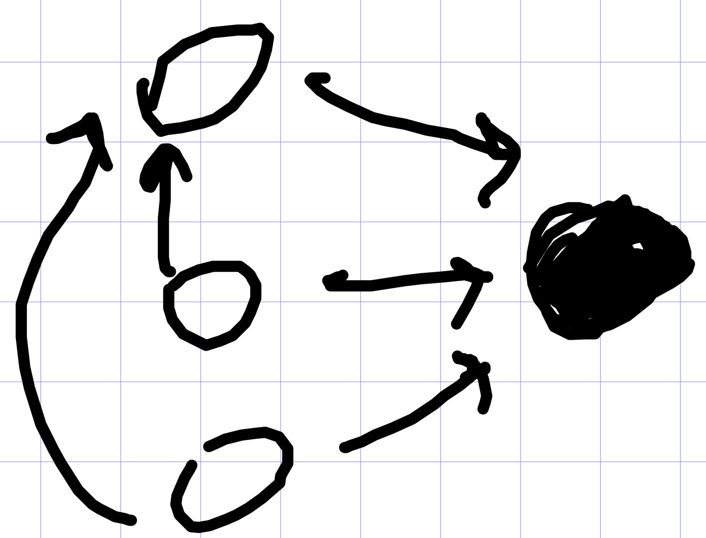
```
Sink-test (G, t : vertex) : boolean;
    j ← 1
    while (j <= n) and G[t,j] = 0
        j++
    if G[t,j] = 1 then return false
                  else return true
```

$O(|V|)$

Questo alg controlla se un dato vertice t è un sink (non sink universale)

SINK UNIVERSALE $O(|V|)$





SINK UNIVERSALE $\Rightarrow$ SINK

SINK $\not\Rightarrow$ SINK UNIVERSALE

SINK <u>UNIVERSALE</u> si trova in $O(|V|)$

$$G(i,j) = \begin{cases} 1 & \text{vertice } i \text{ } \underline{\text{non}} \text{ è sink} \\ 0 & \text{vertice } j \text{ } \underline{\text{non}} \text{ è sink} \end{cases}$$

$i \neq j$

IDEA per trovare il sink universale in $O(|V|)$

```
TROVA-SINK-UNIVERSALE (G)
  // ∄ . ∃! sink universale //
  i ← 1; j ← 1;
  while (j ≤ n) AND (i ≤ n) do
   { if G[i, j] = 1 then i++
                    else j++
   }
  if i > n then return ∄
           else test-sink (G, j)
           // controllo riga e colonna j //
```

L'alg appena proposto non funziona per trovare il sink (funziona solo per il sink UNIVERSALE)

Nel caso del sink $M[i,j] = 0 \quad i \neq j$ non esclude il vertice colonna j.

Componenti fortemente connesse (G)

(si applica ad un grafo G orientato,

DFS (G) with time

Calcola G^T

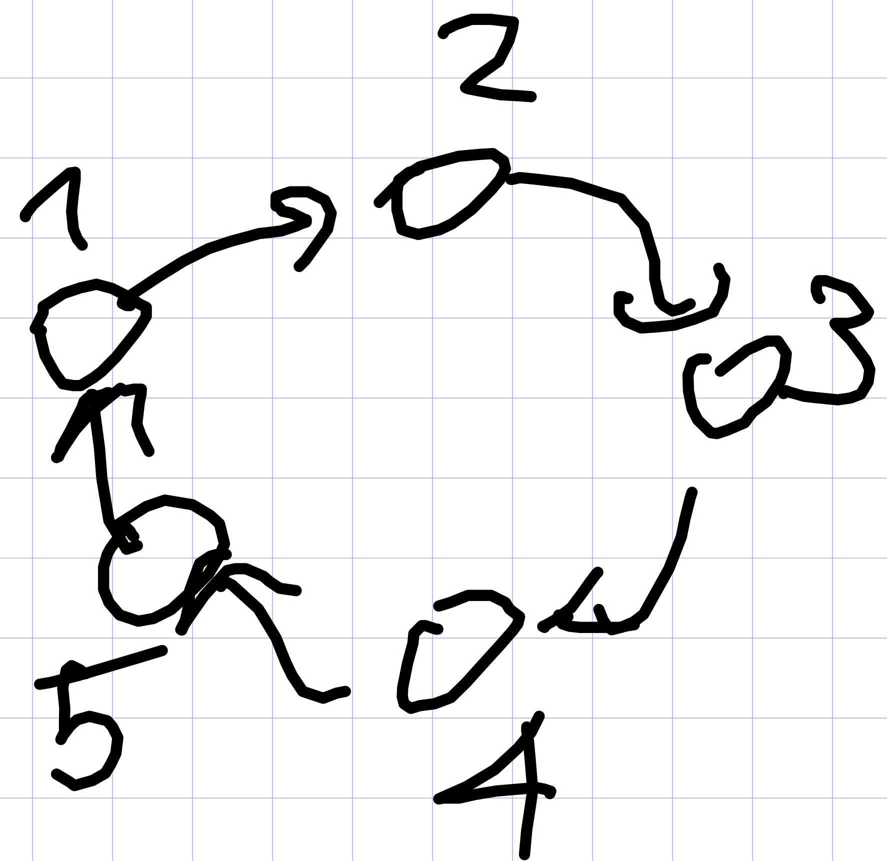
DFS (G^T) chiamando DFS-visit su

vertici con ordine decrescente

di tempo di fine visita

della prima DFS (G)

1 solo albero di copertura



$\forall (u, v) \in V^2$

$u \rightsquigarrow v$

5
4
3
2

DFS.visit(G, 1)

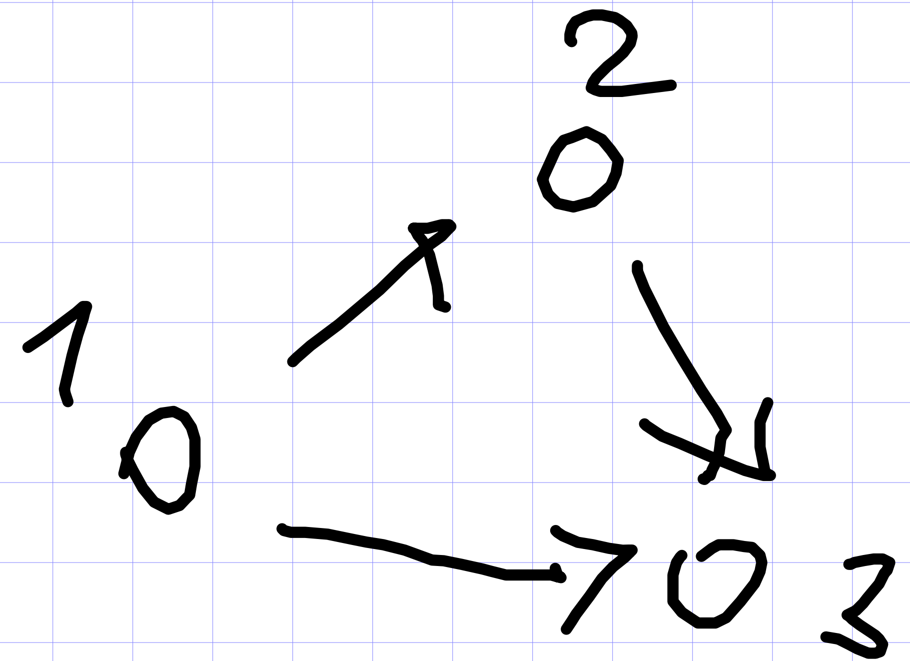
{1 2 3 4 5}

2 4 → 2 3 4

4 2 → 4 5 1 2

3 5 → 3 4 5

5 3 → 5 1 2 3



$1 \rightarrow \{1, 2, 3\}$

$2 \rightarrow \{2, 3\}$

$3 \rightarrow \{3\}$

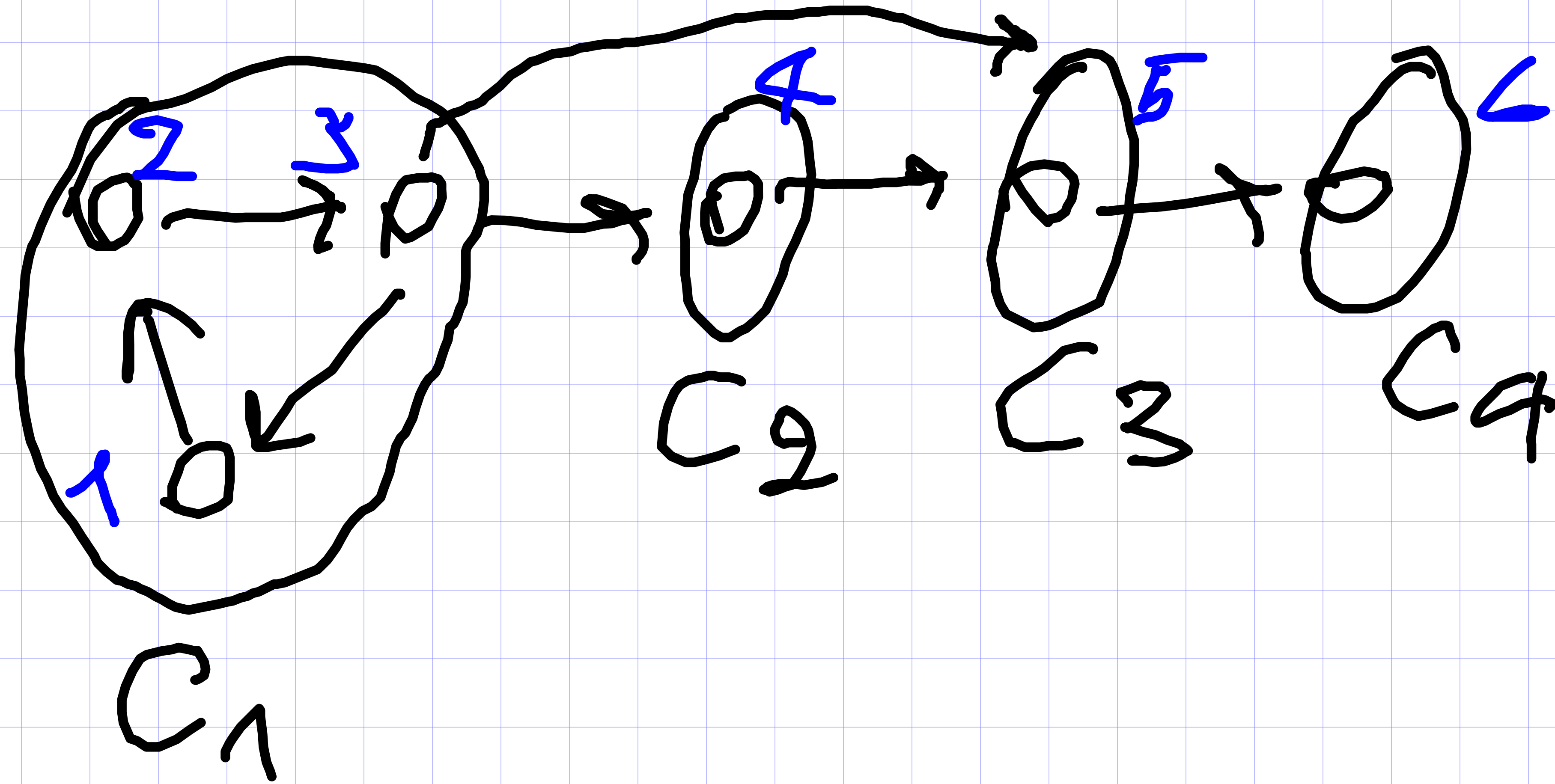
DFS(G, 1) 1T

DFS(G, 2) 2T

DFS(G, 3) < 2T, 3T

DFS(G, 1)

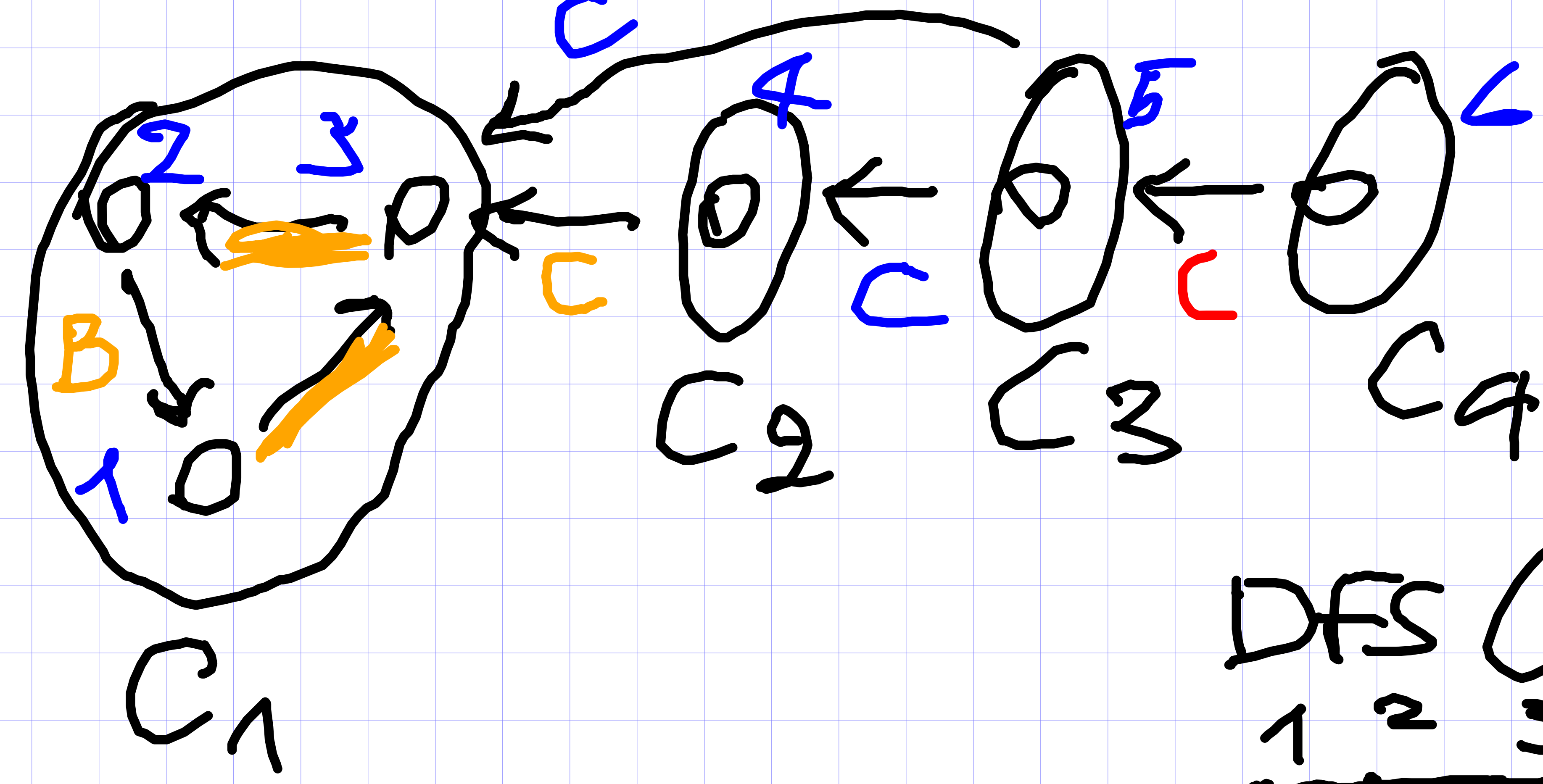
DFS(G, 2)



DFS(G)

	1	2	3	4	5	6
color	~~g~~ b	~~g~~ b	~~g~~ b	~~g~~ b	~~g~~ b	~~g~~ b
π	nil	1	2	3	4	5
d	1	2	3	4	5	6
f	12	11	10	9	8	7

G^T



4 c.f.c

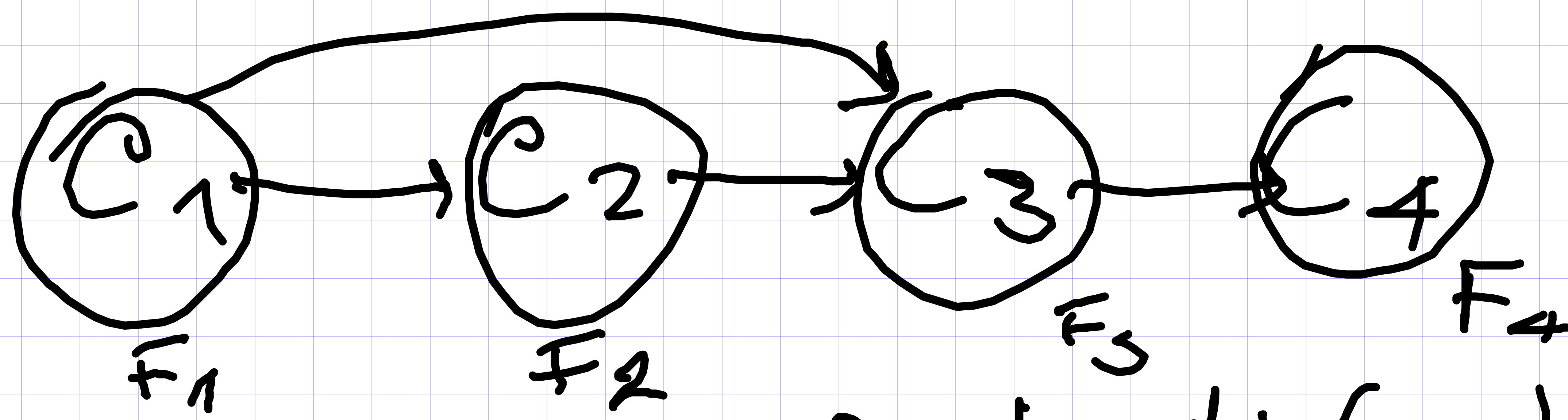
DFS(G)

	1	2	3	4	5	6
color	~~g~~ b	~~g~~ b	~~g~~ b	~~g~~ b	~~g~~ b	~~g~~ b
π	nil	1	2	3	4	5
d	1	2	3	4	5	6
f	12	11	10	9	8	7

DFS(G^T)

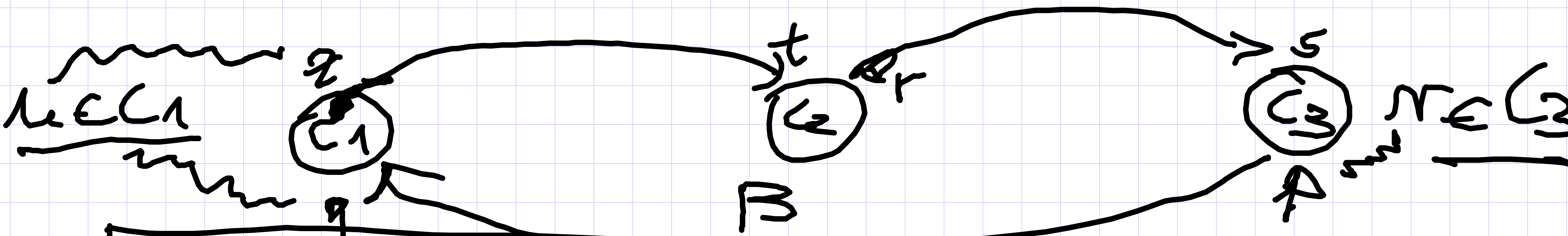
1	2	3	4	5	6
g	g	g	g	g	g
~~3~~	3	1	∅	∅	∅
1	3	2	7	9	11
6	4	5	8	10	12

grafo delle componenti fortemente (c.f.c) connesse G^{SCC} è un DAG



$$G^{SCC} = \{ \underset{\substack{\uparrow \\ \text{un nodo per ogni c.f.c.}}}{V} ; \overset{\substack{\text{archi di } G \text{ che} \\ \text{connettono c.f.c.} \\ \swarrow}}{E} \}$$

G^{scc} D.A.G



$$\boxed{C_1 \cup C_2 \cup C_3 \text{ è una c.f.c}}$$

$$\boxed{C_1 \cup C_3 \ \bar{\text{e}} \text{ una c.f.c}}$$

$\forall\, u, v : \underline{u \in C_1},\ v \in C_3$

$u \rightsquigarrow v$

$v, u : v \in C_3,\ u \in C_1$

$v \rightsquigarrow u$

C e C' siano due c.f.c

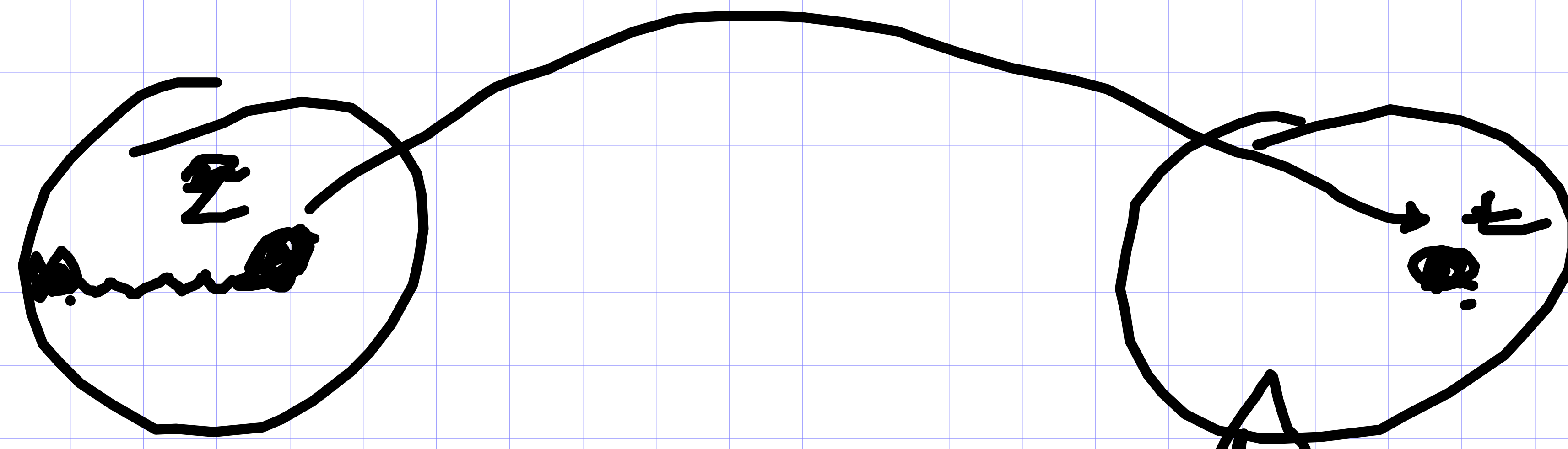
$\exists\ u \to v$ con $u \in C$ e $v \in C'$

$\Rightarrow\ \bar{f}_C > \bar{f}_{C'}$

$\bar{f}_C$ è il max fine visita in C

→ Se DFS parte da un vertice in C esplora C' e poi chiude il vertice in C. (arco congiunzione $\in T$)

→ Se parte da un vertice in C' esplora C' e poi riprende da C. (arco conf $\in C$!)



C $\qquad\qquad$ C'

$F_C > F_{C'}$

DFS di un vertice in C' $\Rightarrow$ esploro C' e prendo da C

Se calcolo G^{T}_{scc} ottengo un "REVERSE SORT TOPOLOGICO"

G archi in blu

$F_5 > F_4 > F_3 > F_2 > F_1$

ogni albero di copertura rappresenta una c.f.c.

Complessità

G^T

G

$$\begin{array}{cc} & \begin{array}{cccc} & & & 4 \end{array} \\ \begin{array}{c} \\ \\ 2 \\ \\ \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

G^T

$$\begin{array}{cc} & \begin{array}{cccc} & & 3 & \end{array} \\ \begin{array}{c} \\ \\ \\ 4 \end{array} & \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

M.A.

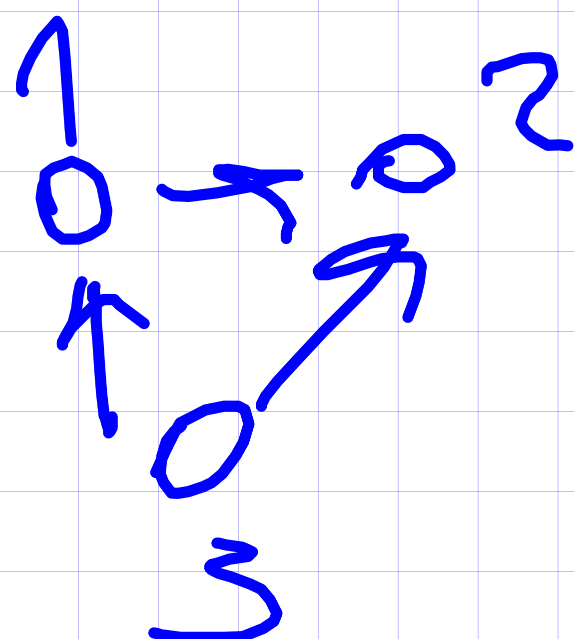
for $i \leftarrow 1$ to $|V|$ do
    for $j \leftarrow 1$ to $|V|$ do
        $G^T[i,j] \leftarrow G[j,i]$

$O(|V|^2) \Rightarrow O(|V|^2)$

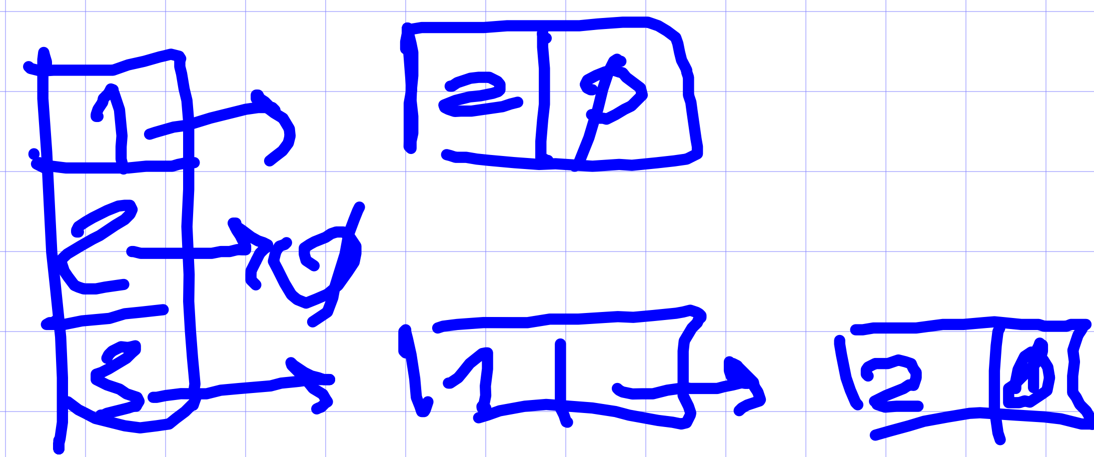
[illegible] $O(|V| + |V|^2)$

Congente'

L.A.



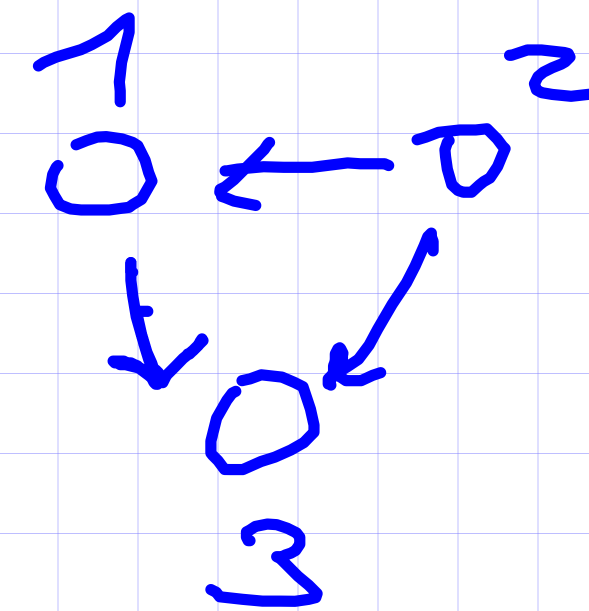
G

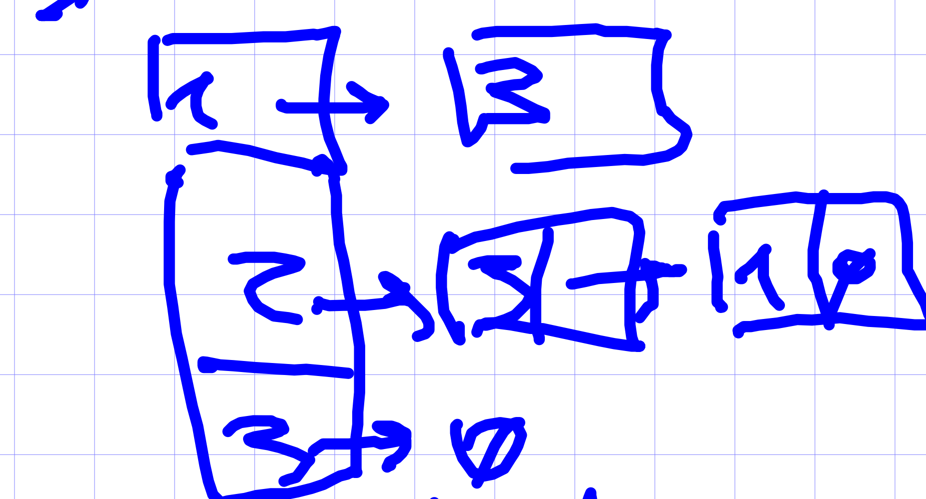


G^T

ins fondo
meno efficiente

G^T





ins testa
(V+E)

O(V+E)
esplorazione
O(V+E)