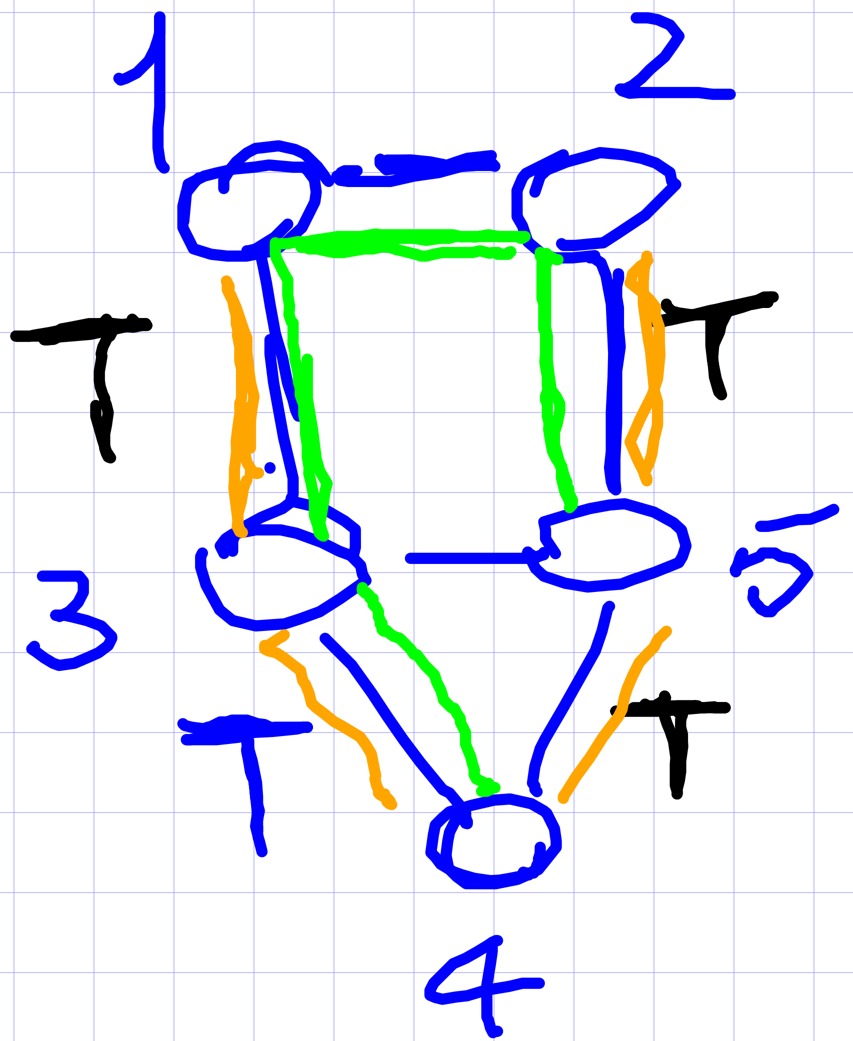
Breadth first search



— BFS

— DFS

	1	2	3	4	5
col	b	g	b	b	b
π	3	5	4	∅	4

source 4

Q ~~4~~ | ~~1~~ | ~~3~~ | ~~5~~ | ~~1~~ | ~~2~~ |

BFS(G ; s : source)

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

$\pi(v) \leftarrow$ nil

color(s) $\leftarrow$ gray;

ENQUEUE(Q, s);

while ($Q \neq \{\emptyset\}$) do

$u \leftarrow$ Dequeue(Q)

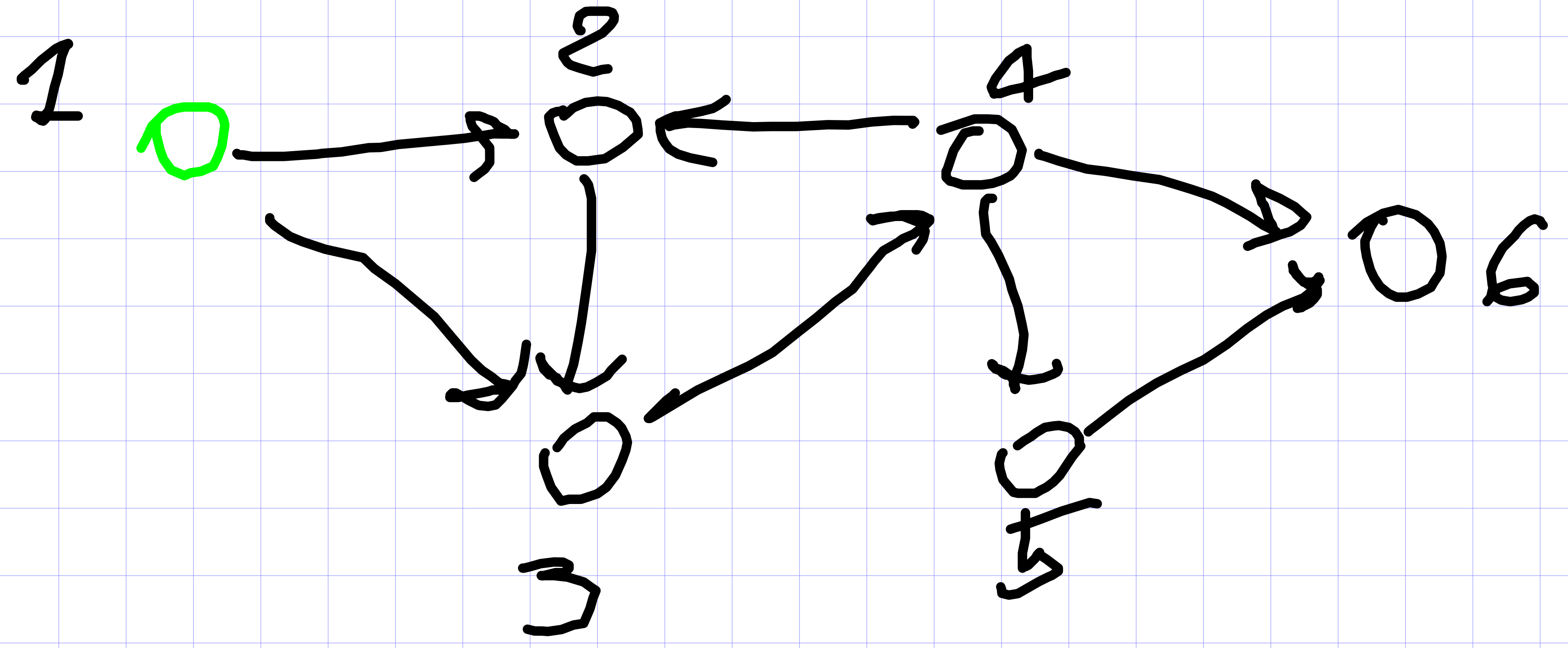
$\forall v \in Adj(u)$ do

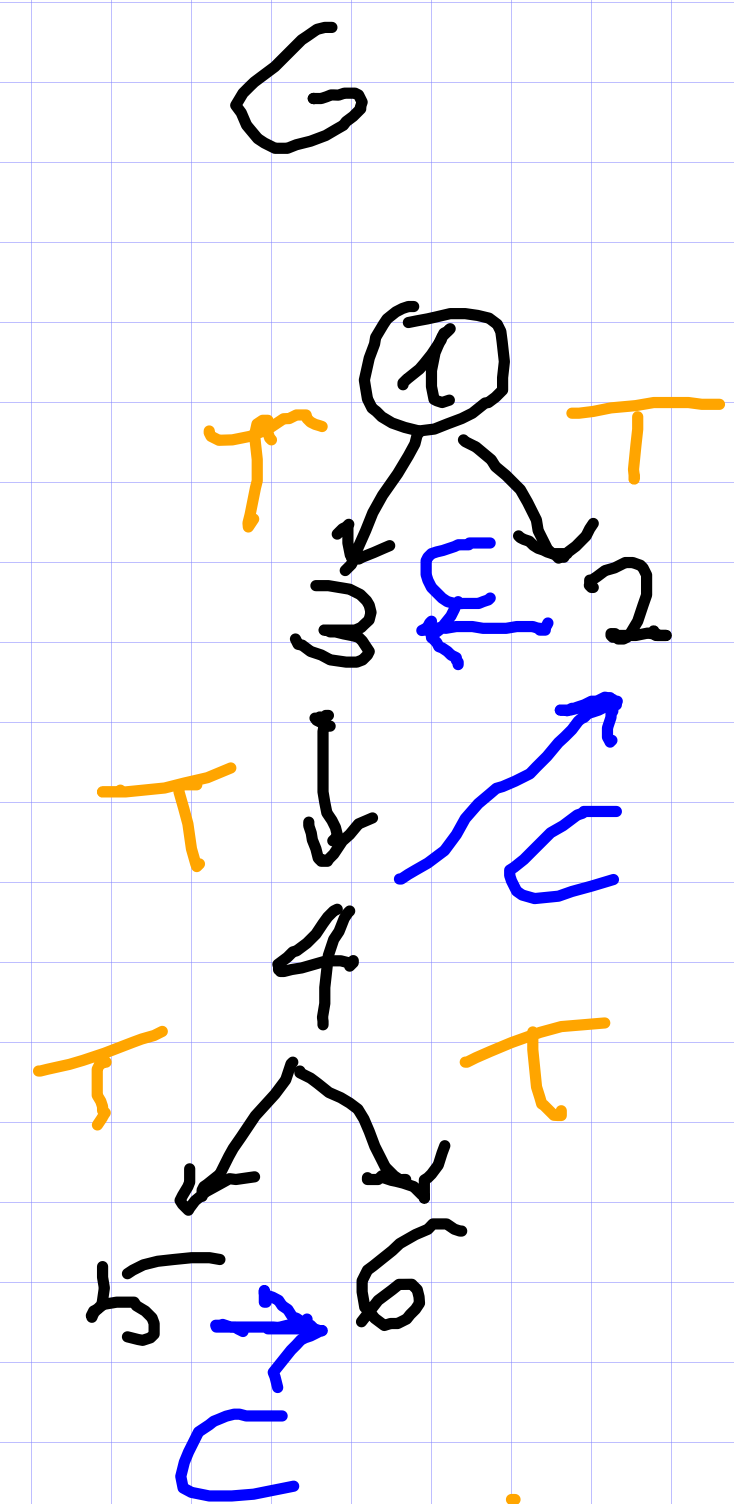
if color(v) = white then

$(u,v) \in T$, $\pi(v) \leftarrow u$; color(v) $\leftarrow$ gray;

ENQUEUE(Q, v)

color(u) $\leftarrow$ black





BFS (G, 1)

	1	2	3	4	5	6
color	b	b	b	b	b	b
π	Ø	1	1	3	4	4

Q	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~

ORIENTATO

{T, C, B}

F

BFS $(G = (V, E);\ s \in V)$ con calcolo delle distanze

$\forall v \in V - s$ { color$(v) \leftarrow$ white
$\pi(v) \leftarrow$ nil
$d(v) \leftarrow +\infty$
}

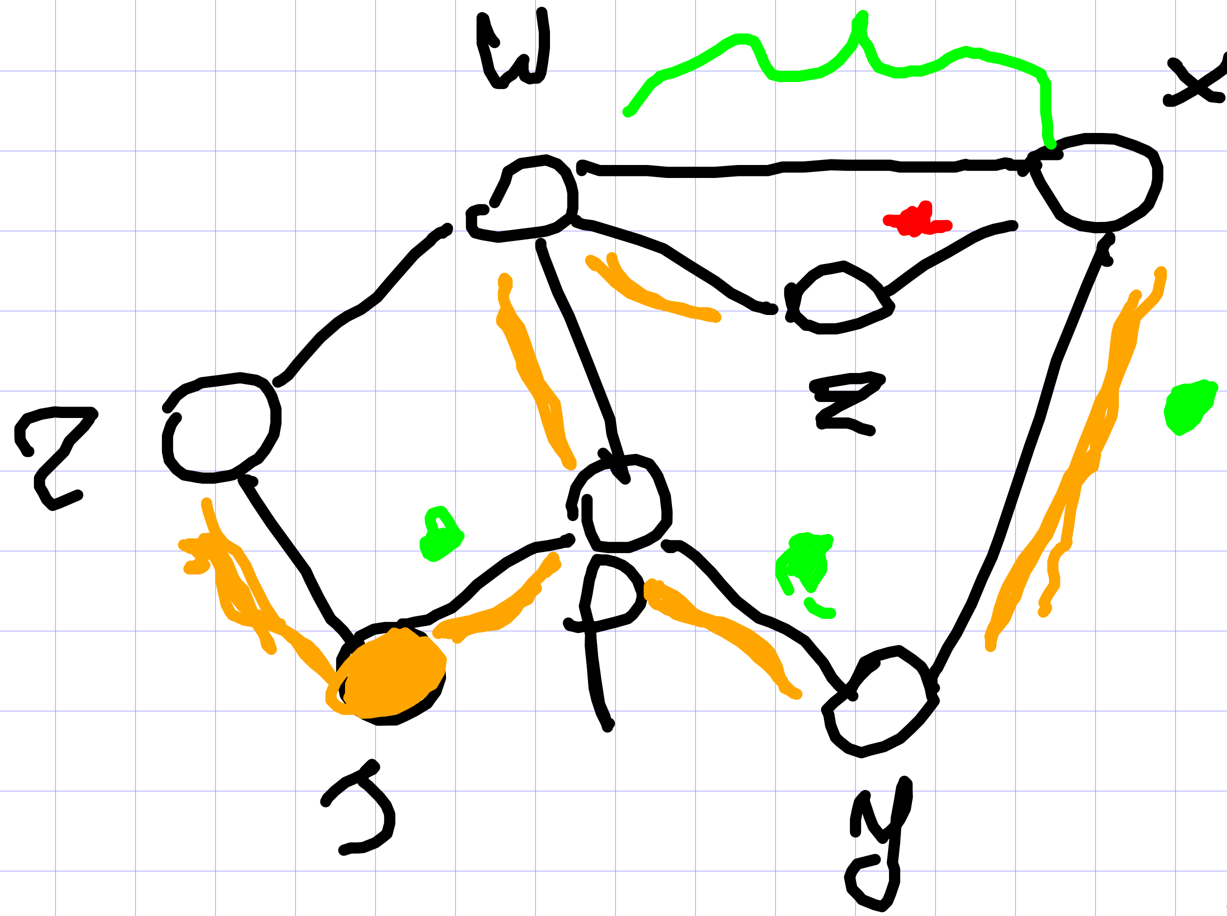
color$(s) \leftarrow$ gray. $d(s) \leftarrow 0$; $\pi(s) \leftarrow$ nil

$Q \leftarrow \{\emptyset\}$

ENQUEUE(Q, s)

while $Q \neq \{\emptyset\}$ do
$u \leftarrow$ DEQUEUE(Q)
$\forall v \in$ Adj$[u]$
if color(v) = white then
color$(v) \leftarrow$ gray; $\pi(v) \leftarrow u$
$d(v) \leftarrow d(u) + 1$;
ENQUEUE(Q, v)

color$(u) \leftarrow$ black



	s	r	p	w	z	x	y
colore	b	b	b	b	b	b	b
d	$\cancel{0}$	1	1	2	3	3	2
π	nil	s	s	p	w	y	r

$d(v)$ = distanza da s a v
= minimo numero di archi da s a v
= shortest path da s a v

BFS (G, s)

Q $\boxed{\cancel{s} \mid \cancel{p} \mid \cancel{r} \mid \cancel{y} \mid \cancel{w} \mid \cancel{x} \mid \cancel{z}}$

→ $d(v)$ dipende solo dalla sorgente

→ d è crescente

→ BFS (G, s) visita ogni vertice raggiungibile da s.

→ $\boxed{\delta(s,s) = \phi = d(s)}$

$\delta(s, v)$ = distanza da s a v

$\delta(s, v) = d(v)$

$\rightarrow \forall e = (u, v) \in E$

$$\delta(s, v) \leq \underbrace{\delta(s, u) + 1}$$

$s \rightsquigarrow^{*} v$

$s \rightsquigarrow u, (u, v) \in E$

L'arco $e = (u, v)$ implica l'esistenza di un cammino da s a v. Tale cammino ha lunghezza $\delta(s, u) + 1$ e pertanto o è lo shortest

$$\delta(s, v) = \delta(s, u) + 1,$$

o è più lungo

$$\delta(s, u) + 1 > \delta(s, v)$$

$\forall v \in V$

(1) $d(v) \geq \delta(s, v)$ durante l'algoritmo

Inizialmente $d(v) = +\infty$ e ogni shortest path ha lunghezza finita

Poi ragioniamo per induzione.

Base induzione vertice source s. $d(s) = 0 = \delta(s, s)$

Consideriamo il nodo v, supponiamo per induzione che (1) vale per i nodi raccolti

$$d(v) = d(u) + 1$$

$$\geq \delta(s, u) + 1 \quad \text{(per induzione!)}$$

$$\geq \delta(s, v) \quad \text{(per la proprietà precedente!)}$$

Organizziamo la coda Q $\boxed{Q \text{ è MIN-HEAP}}$

Q contiene vertici a distanze crescente da s.

head ... tail

Q

v_h		v_i		v_r

$$d(v_r) \leq \overbrace{d(v_h)}^{\text{testa}} + 1$$

$$\rightarrow d(v_i) \leq d(v_{i+1}) \qquad \forall v_i \in Q$$

Supponiamo che sia vero e modifichiamo

Q : (1) estrazione

quindi $d(v_{h+1}) \geq d(v_h)$

$$d(v_r) \leq d(v_h) + 1 \leq \overbrace{d(v_{h+1})}^{\text{nuova testa}} + 1 \quad \text{ok}$$

② inserzione (v_{r+1}) (v_{k-1}) v_k v_r v_{r+1}

$u = d(v_1)$ subito dopo la rimozione di u

al tempo della dequeue $d(v_{k-1}) \le d(v_i) \ \forall v_i \in Q$

$u \in$ DEQUEUE(Q) $\underbrace{d(u)} \le \underbrace{d(v_k)}$ nuova testa

$$d(v_{r+1}) = d(u) + 1 \le d(v_k) + 1$$

Quindi ne consegue che i vertici in coda hanno etichette crescenti

Si noti anche che un arco (u,v) verso un vertice $\boxed{\text{grigio}}$ nella coda è un arco verso un vertice che si trova alla stessa distanza di u da s, $d(v) = d(u)$, oppure a distanza $d(v) = d(u) + 1$

$$\begin{cases} d(v) = d(u) + 1 \\ d(v) = d(u) \end{cases}$$

$\forall v \quad d(v) = \delta(s, v)$

Ragioniamo per induzione $\quad \bar{u} = \pi(v)$

$$d(s) = \delta(s, s) = 0$$

Sia v il <u>primo</u> vertice per cui

(1) $\quad d(v) > \delta(s, v) = \delta(s, \bar{u}) + 1 \quad$ [Eq. 1]

$$= d(\bar{u}) + 1$$

Al tempo $\bar{u}$ è stato estratto da Q

v poteva essere

nero: $\quad d(v) < d(\bar{u})$ FALSA (1)

gray: $\quad d(v) \leq d(\bar{u}) + 1$ FALSA (1)

white: $\quad d(v) = d(\bar{u}) + 1$ FALSA (1)

In tutti i casi Eq. 1 è FALSA

$\Rightarrow$ falsa l'assunzione che $\exists\, v \;\; d(v) \neq \delta(s, v)$

Modifica BFS per restituire il nodo più lontano dalla sorgente s.

```
BFS (G, s) : integer
  ∀ v ∈ V - s  do   d(v) ← +∞, π(v) ← nil
                    color (v) ← white
  d(s) ← 0; π(s) ← nil; color (s) ← gray
  ENQUEUE (Q, s)
  while (Q ≠ ∅) do
      u ← DEQUEUE(Q)
       ∀ v ∈ Adj(u) do
            if color(v) = white then
                 d(v) ← d(u)+1; π(v) ← u
                 ENQUEUE(Q, v)
                 last ← d(v)
      color (u) ← black
  return last
```

Modifica BFS che non usa l'attributo COLOR

BFS (G, s);
$\forall v \in V - s$ do $\{ d(v) \leftarrow +\infty, \pi(v) \leftarrow nil \}$

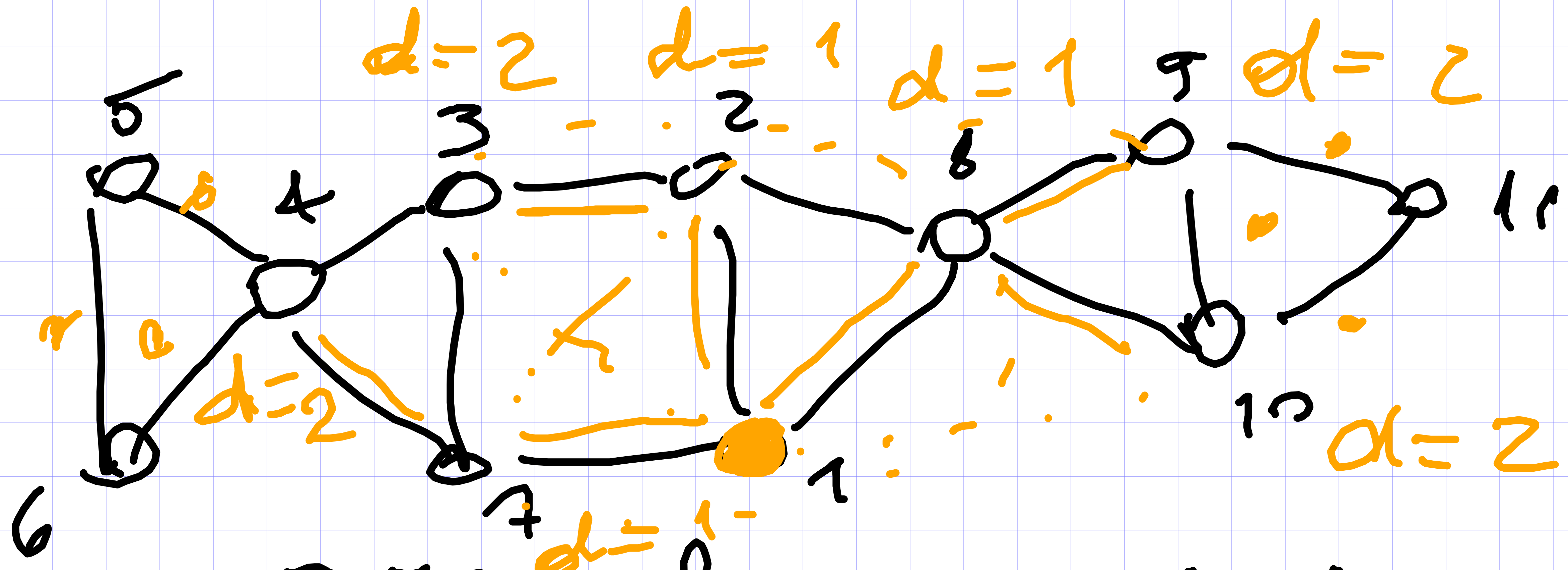
$d(s) \leftarrow \emptyset$; $\pi(s) \leftarrow nil$;
ENQUEUE (Q, s)
while $(Q \neq \emptyset)$ do
  $u \leftarrow$ DEQUEUE(Q)
  $\forall v \in Adj(u)$ do
    if $(d(v) = +\infty)$ then
      $\{ d(v) \leftarrow d(u) + 1$; $\pi(v) \leftarrow u$
      ENQUEUE$(Q, v) \}$

Modifica BFS per restituire i nodi a distanza $\leq t$ dalla sorgente

vertici $= |N_t(s)|$

```
BFS(G, s)
  ∀ v ∈ V - s do  d(v) ← +∞, π(v) ← nil
                  color(v) ← white
  d(s) ← 0; π(s) ← nil; SOL ← ∅
  ENQUEUE(Q, s)
  while (Q ≠ ∅) do
      u ← DEQUEUE(Q)
      ∀ v ∈ Adj(u) do
          if color(v) = white then
  SOL ← SOL ∪ v;  d(v) ← d(u) + 1; π(v) ← u;
              if (d(v) < t) then
                    ENQUEUE(Q, v)
      color(u) ← black
  return SOL
```

$$N_t(s) = \{ v \in V : d(v) = \delta(s, v) \leq t \}$$



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

BFS fino a distanza $2 = t$

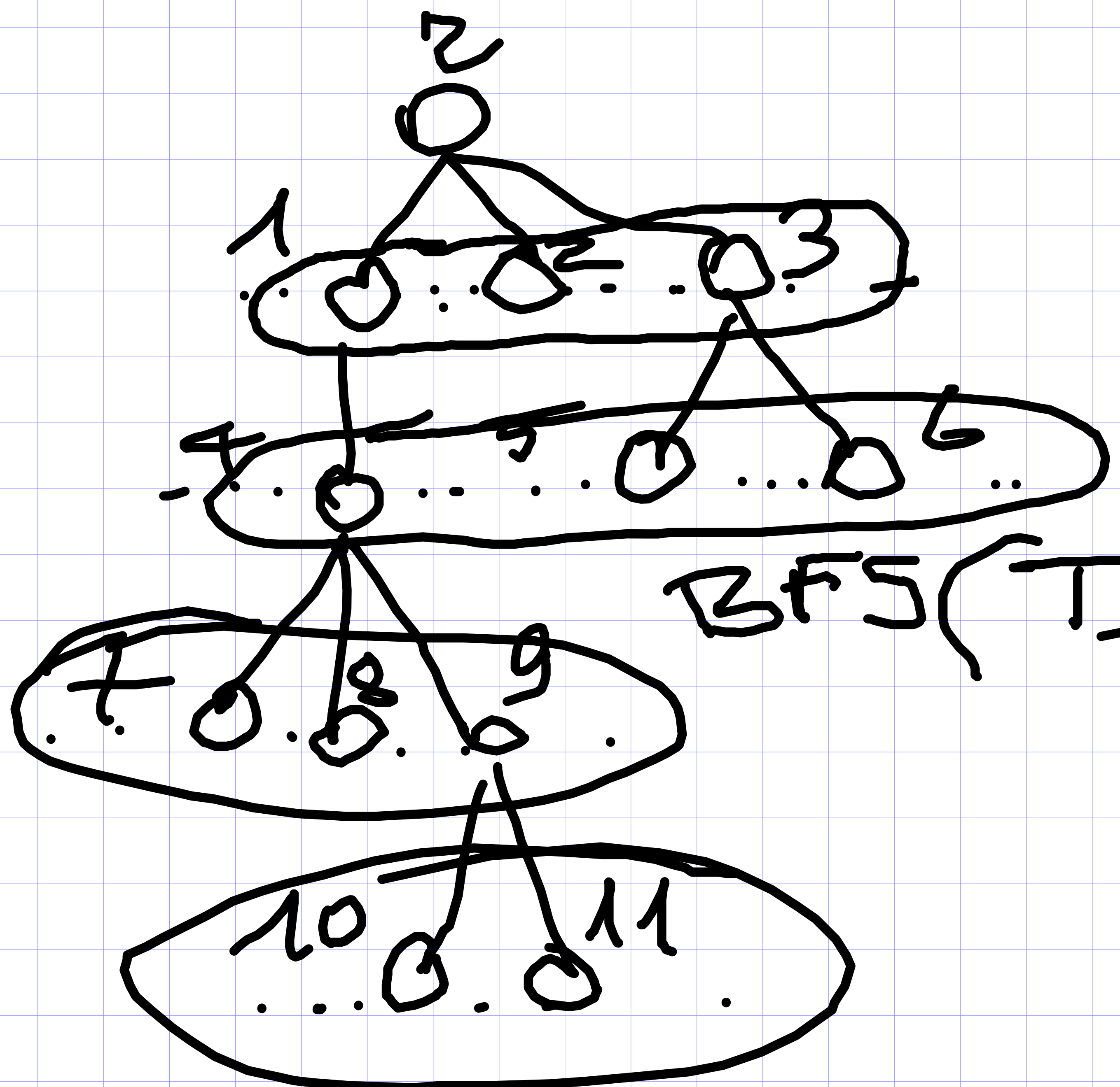
	b	b	b	b
Q	1	2	7	8
d(Q)	0	1	1	1

$V' = |N_2(1)| = 8$

$E' = \min\{E, V'^2\}$

$Sol \leftarrow \{1, 2, 7, 8, 3, 4, 9, 10\}$

Dato albero T fare visita per livelli di T.



BFS(T, r)

$r \quad d(r) = \emptyset$

$v : \quad d(v) = 1$

$v : \quad d(v) = 2$

$v : \quad d(v) = 3$

$v : \quad d(v) = 4$

= visitare i vertici in ordine crescente di distanza dalla radice diametro $(G) = \{ u, v \in V^2 : \delta(u,v)$ è massimo

ossia $\delta(u,v) \geq \delta(p,r) \; \forall \, p, r \in V^2 \}$

$diam \leftarrow \emptyset$

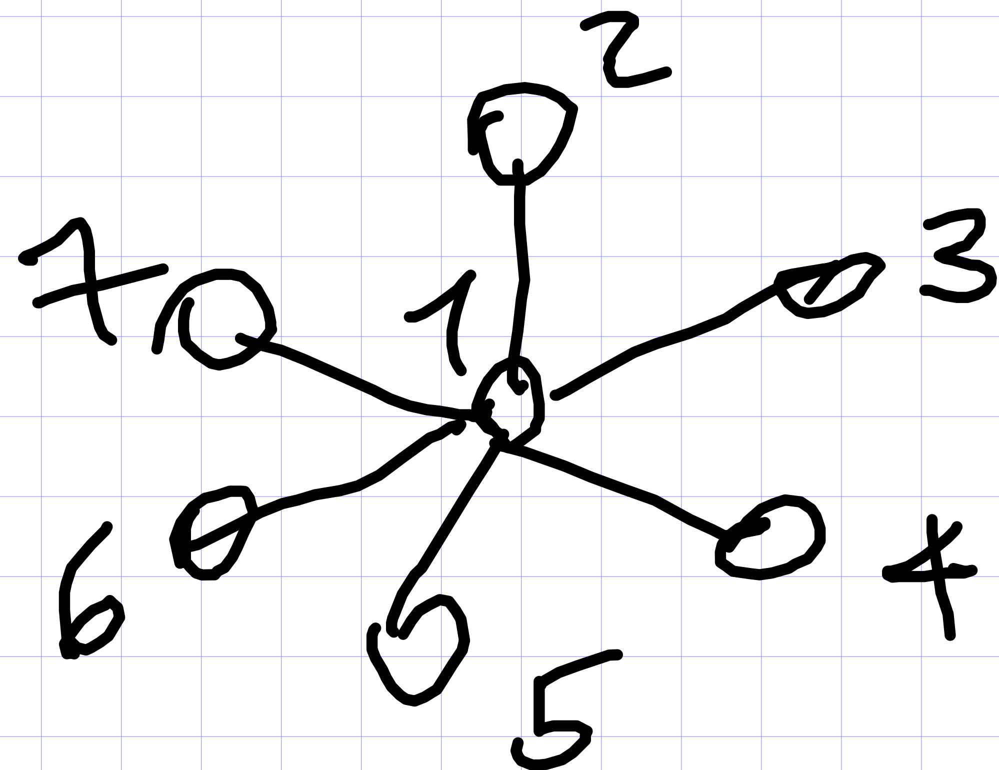
$\forall \, v \in V$

$temp \leftarrow BFS(G, s)$ con last

if $diam \leftarrow temp$ then update diam

Complessità $V(V+E)$ $V(V^2)$

G



$\mathrm{diam}(G) = 2$

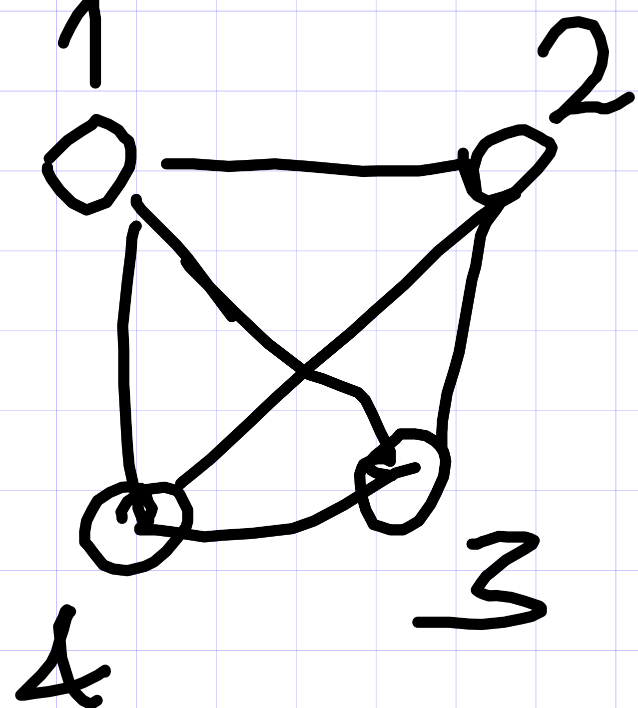
$d(2,3) = 2$

$d(1,x) = 1$

$d(i,j) = 2$

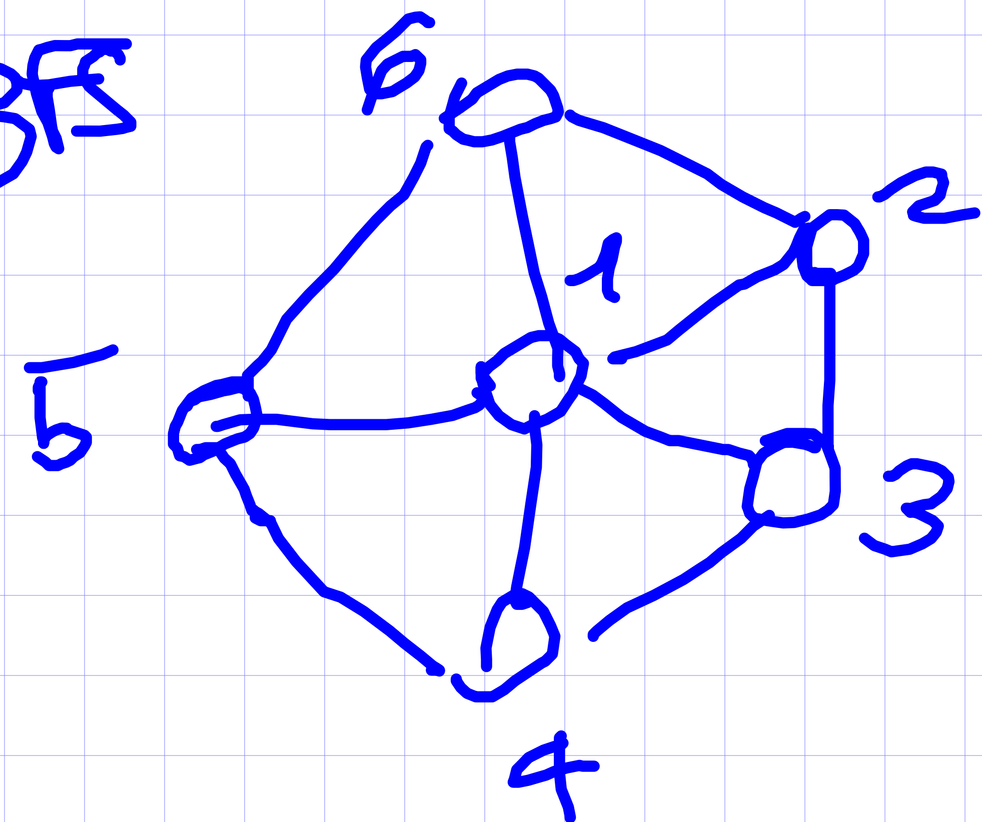
$i \neq j \neq 1$

K_4



$\mathrm{diam}(K_4) = 1$

Dato un grafo G, trovare il diametro di G



11
12
13
14
15
16
} BFS

21
22
23
24
25
26

$max \leftarrow \phi$

$\forall\ u, v : V \times V$

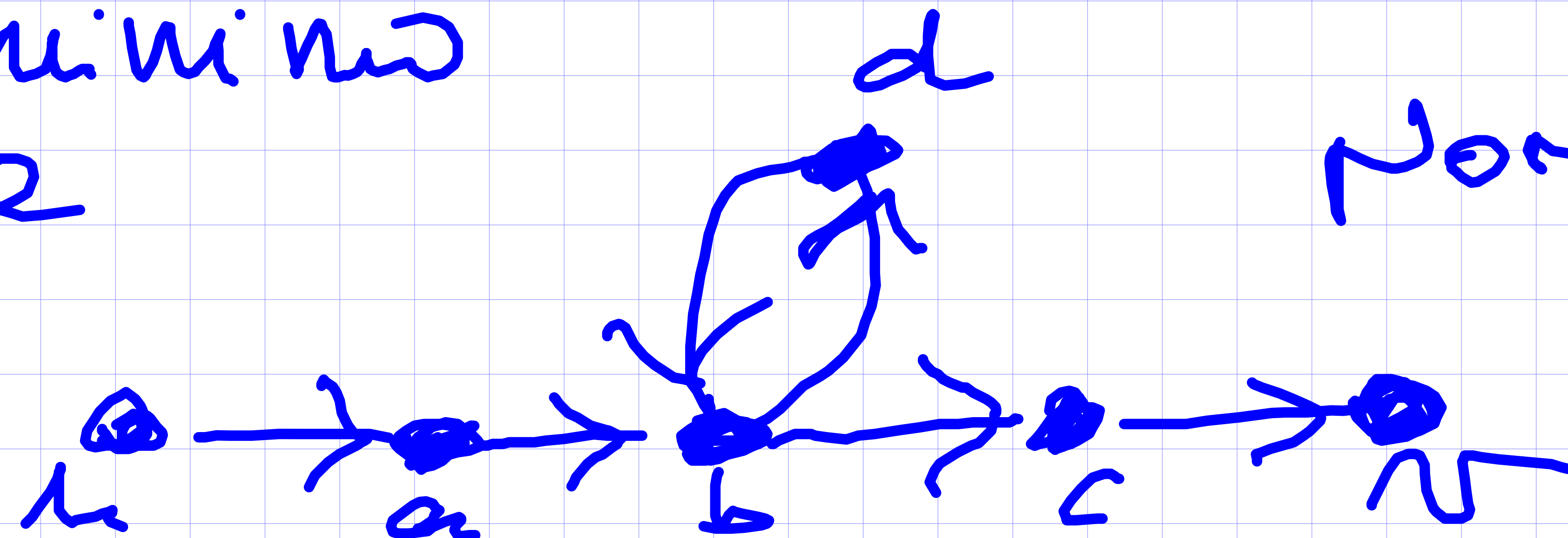
if $d(u, v) > max$ then

$max \leftarrow d(u, v)$

BFS(G, u) troncata quando v è ENQUEUED

return max

cammino minimo
non ha
cicli



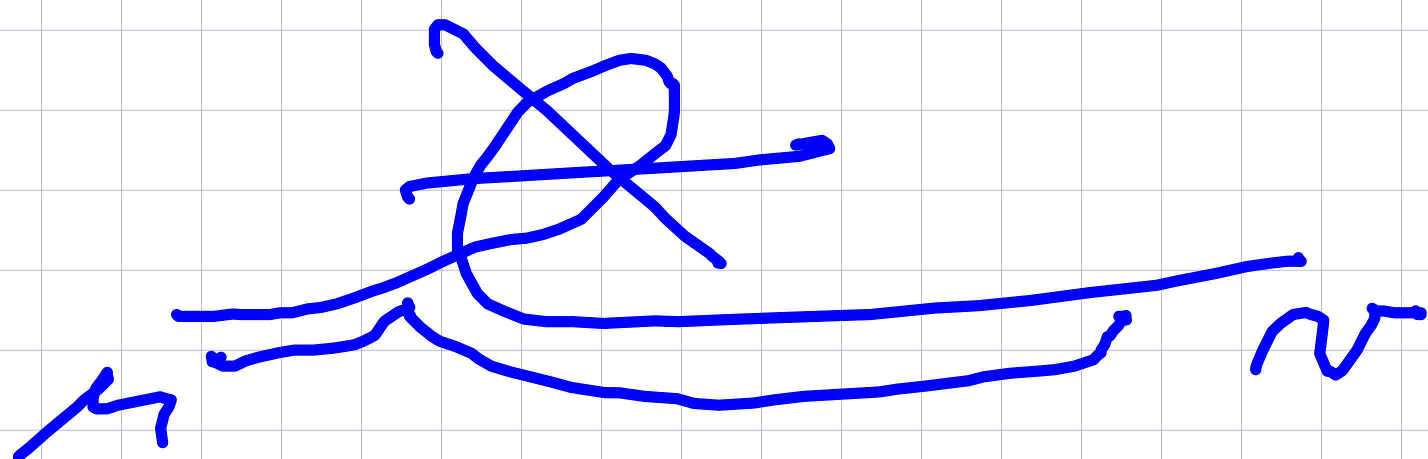
Non è cammino
MINIMO
da u a v

$u \to a \to b \to c \to v$

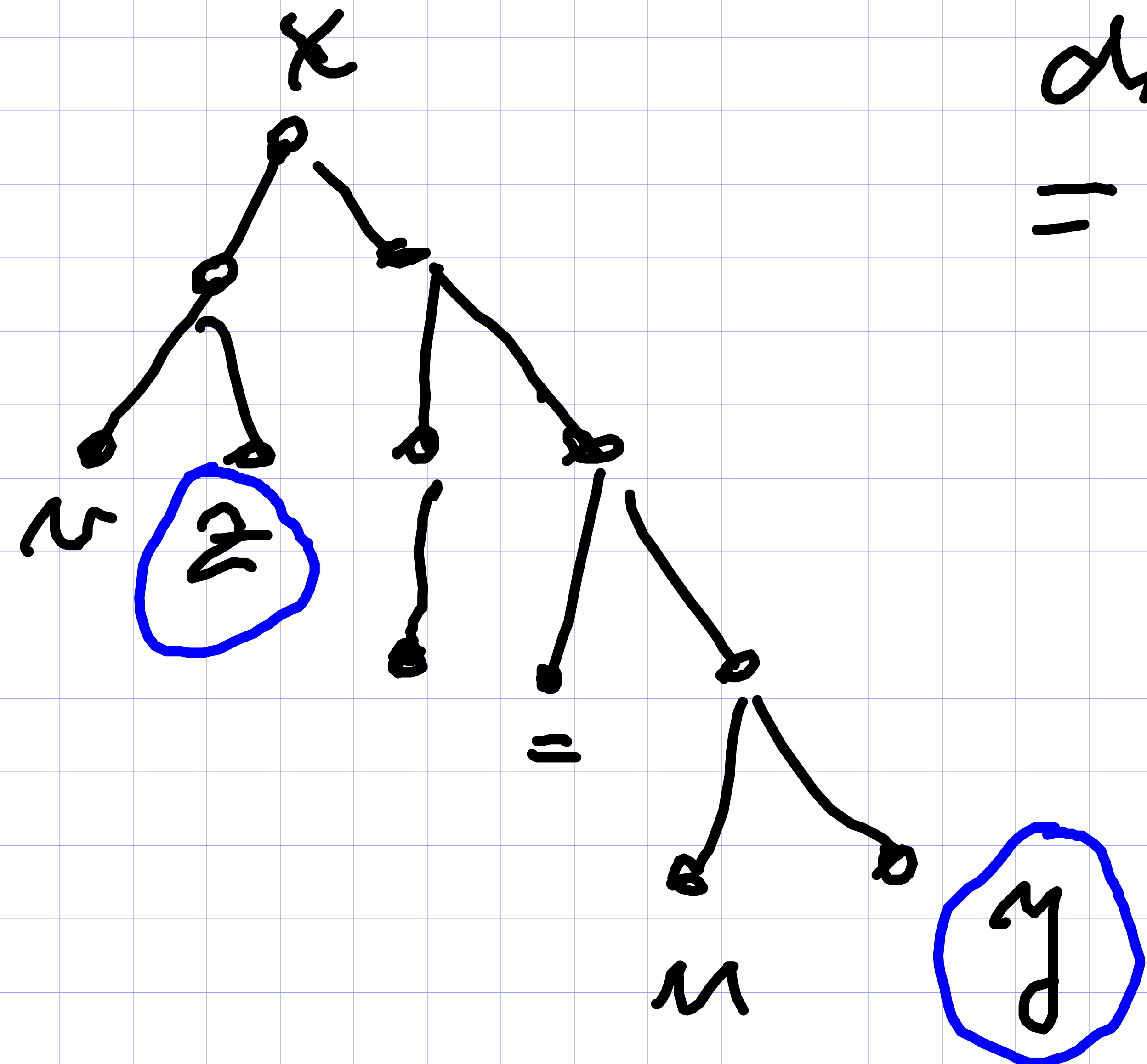
n vertici

$1 \leq \text{diam}(G) \leq n-1$

$u - \circ - \circ - \circ - v$



l'ultimo vertice scoperto BFS è il vertice più lontano da x.



$\delta(x, y)$ = max distanza da x

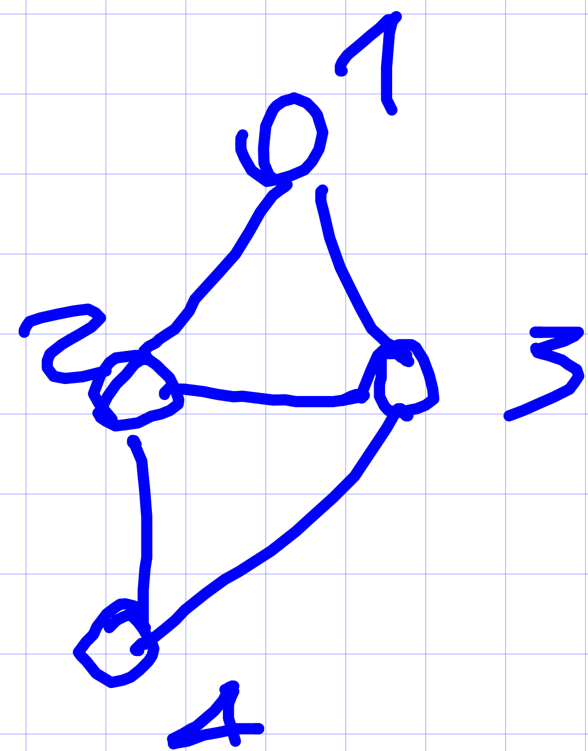
diametro$(T) = \delta(y, z)$ dove z è il nodo più lontano da y.

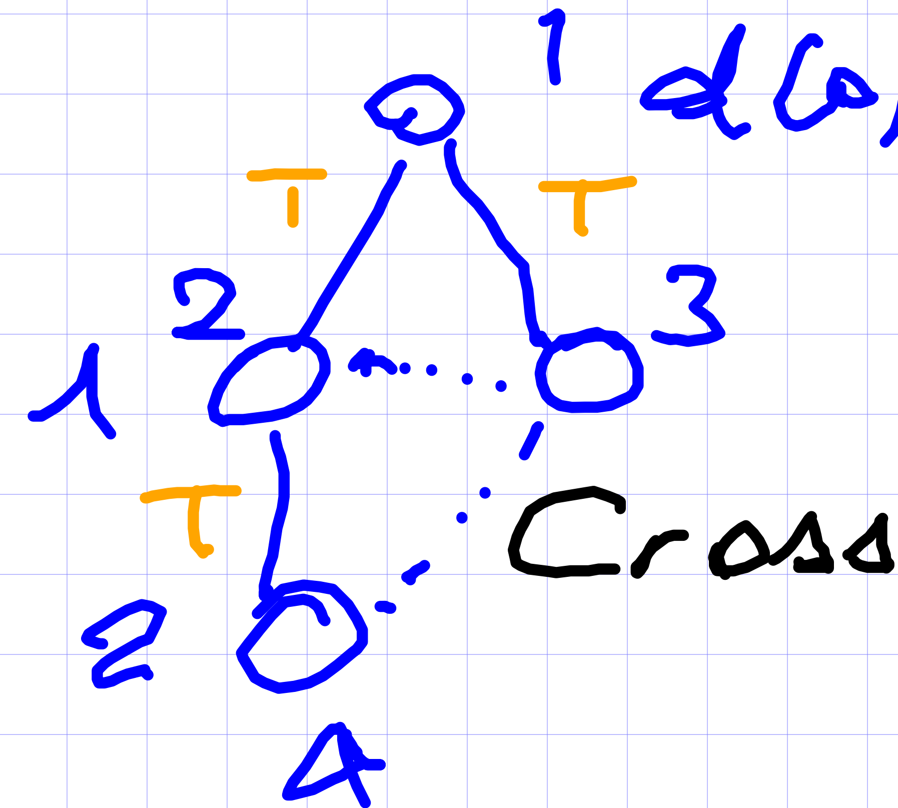
Il diametro di T si calcola in $O(V)$

Archi nella BFS

archi sono denominati rispetto alla loro posizione dei loro estremi in T

Back : verso antenato
Forward : verso discendente
Tree : arco esplorazione
Cross : otherwise





Grafo non orientato

Backward : $\nexists$

Forward : $\nexists$

$(u,v) \in T$: $d(v) = d(u) + 1$ v white

$(u,v) \in C$: $\left.\begin{array}{l} d(v) = d(u) \\ \text{or } d(v) = d(u) + 1 \end{array}\right\}$ $v \neq$ white

Grafo orientato BFS

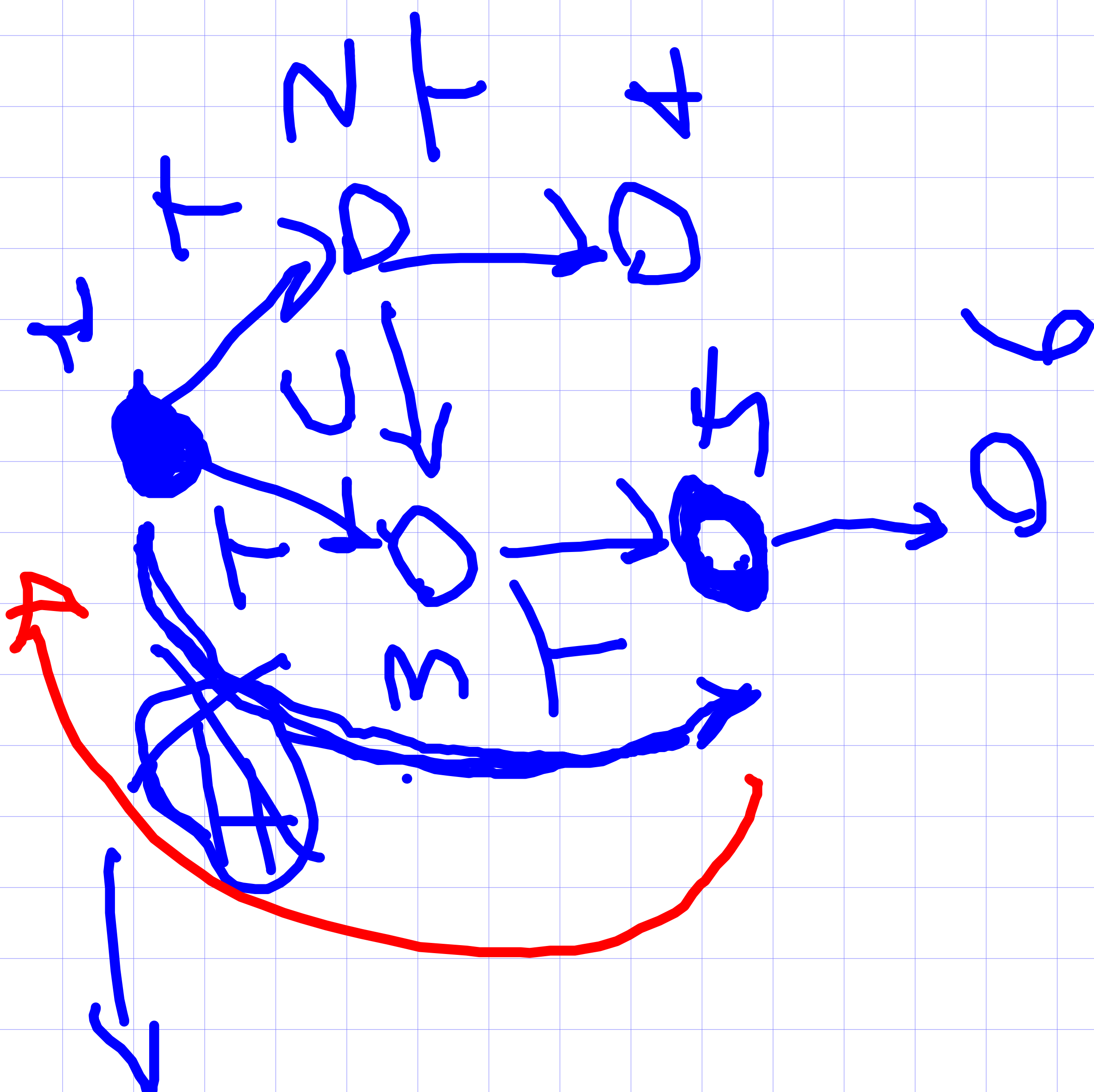
1. no forward edges
2. $(u, v) \in T$ $\quad d(v) = d(u) + 1$

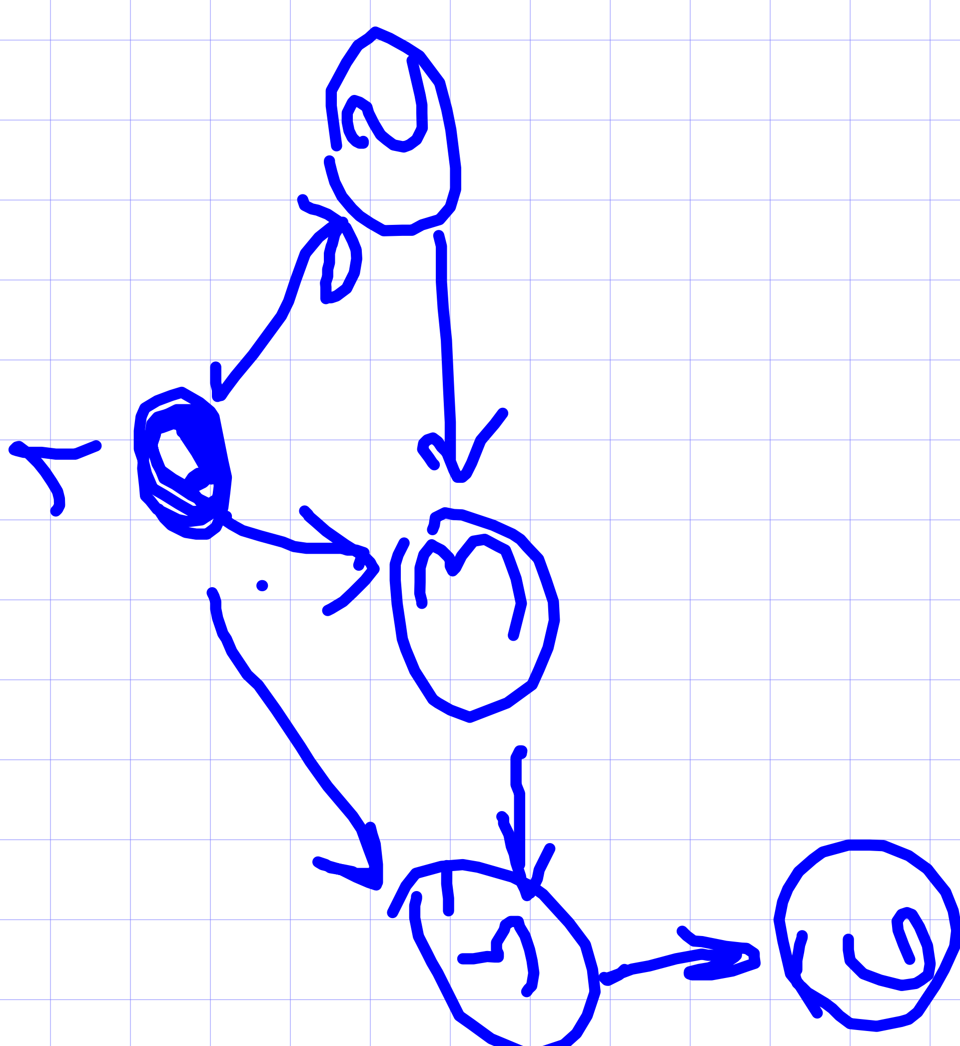
   v white

3. $(u, v) \in C$ $\quad$ né in T

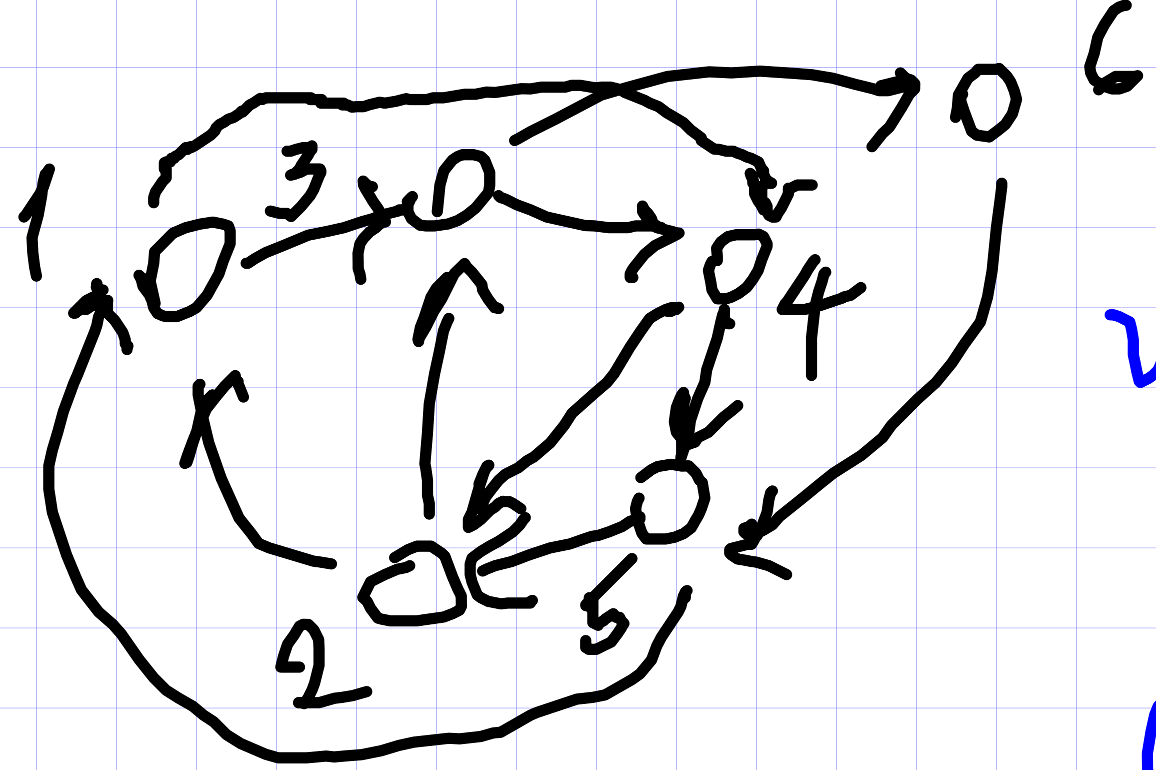
   né in B

4. $(u, v) \in B$ $\quad d(v) < d(u)$





EULERO TOUR : visita tutti gli archi una sola volta



VISIT (G, L, v)

```
C ← ∅
u ← v
while δ+(u) > ∅
    let w ∈ Adj(u);
    delete(u, w)
    δ+(u) --
    add u to C
    if δ+(u) > ∅ add u to L
    u ← w
return C
```

1 4 5 2 1 3 6 5 1

3 4 2 3

```
EULER-TOUR (G)
  T = ∅
  L ← {any vertex in G}
  while L ≠ ∅
      remove v from L
      C = visit (G, L, v)
      if location in T = nil then
                    T ← C
          else introduce C before
               location of v in T
```