ESERCIZIO 1:

Dato un grafo non orientato, scrivere un algoritmo che decide se il grafo é un **ALBERO**.

non orientato, connesso, aciclico.

→ SPEZZIAMO IL PROBLEMA:

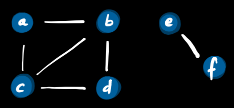
* Dato $G=(V,E)$ NON OR. scrivere un algoritmo che decide se G é **CONNESSO**.

ossia: SSE $\forall u,v \in V$ v é RAGGIUNGIBILE da u.

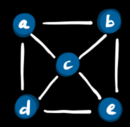
→ RAGIONAMENTO:

Come faccio? prendo un v, faccio $BFS(G,v)$. Se alla fine $\exists v \in V$ t.c. $v.d = \infty$ → allora non ho un grafo conn.

→ altrimenti ho un grafo conn.



"qui ad esempio avrei "False" come risultato



qui ad esempio avrei "true" come risultato.

Piccola definizione di **Componente Connessa**: ogni elemento di $V/\sim$, con $\sim$ relazione di raggiungibilità.

posso raggiungere $u \to v$ e $v \to u$.

ALGORITMO:

```
isConnected (G) =
  s = random(V)
  BFS(G, s)   // versione con la sola distanza.
  For (v in V):
    IF (v.d == ∞):
      return false;
  return true;
```

* SECONDA PARTE: Il grafo é **ACICLICO**?

TEOREMA:

Un grafo non orientato é un albero sse:

* É CONNESSO e ACICLICO

oppure

* É CONNESSO e $\boxed{|E| = |V| - 1.}$

* Piccola modifica a BFS:

Mentre eseguo la BFS nel punto precedente, conto
- NUMERO ARCHI
- NUMERO VERTICI

ossia nel for $i \in Adj[u]$ della BFS → POSSO CONOSCERE ARCHI E VERTICI.

Occhio! in realtá non conto TUTTI I VERTICI DEL GRAFO.

Se passo isConnected(G) → so che ho visitato ogni vertice, quindi → il numero di vertici é quello che

$n = Adj.length/2$ → ossia: $(n = V$!!!$)$

↳ perché é non orientato (HO LE RIPETIZ.).

↱ numero archi, calcolato in BFS.

```
IF (e == n - 1):
  return True;
```

↳ HO UN ALBERO !

ESERCIZIO 2 :

Dato un grafo $G = (V, E)$ **NON ORIENTATO**, dato un $x \in V$, stabilire se la cc contenente x È UN ALBERO.

* Cosa cambia rispetto a prima?

SE $|E| = |V| - 1$

↳ ma questi solo DELLA COMPONENTE CONNESSA CHE INCLUDE x.

Aggiungo un contatore in più → conto i vertici.

u = DEQUEUE (Q)
n_v++ → ogni vertice che estraggo dalla coda lo conto. → POI CONTROLLO E SONO A POSTO!
For v ∈ Adj[u]

ESERCIZIO 3

Dati:

$G = (V, E)$; $s \in V$; $k > 0 \in \mathbb{N} \setminus \{0\}$.

Modificare BFS in modo tale che scopra solo i vertici alla distanza $\leq k$.

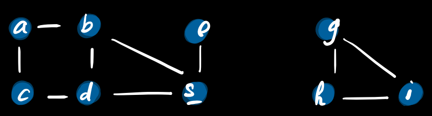
```
DFS_D (G, s, k) :

For (v in V):          // SETTING INIZIALE
    v.color = white;
    v.distance = ∞ ;
    v.parent = ∅ ;

s.color = white ;       // SETTING INITIAL
v.distance = 0;
v.parent = 0;

ENQUEUE( Q, s );        // Inserisco nella queue s.

While ( Q.notempty() ∧ k > 0 ) :   // Finché ho elementi da esplorare e
                                    // ho "risorse" di ampiezza disponibile
    y = DEQUEUE (Q);

    IF ( Adj[y].notEmpty() ):
        k--

    For ( x in Adj(y) ):   // Esploro!
        If ( x.color == "white" ):
            x.color = "gray"
            x.distance = y.distance + 1
            x.π = y;

            Enqueue ( Q, x );
```

y.color = "Black";

ESERCIZIO 4

Dato $G=(V,E)$ NON ORIENTATO ed $s \in V$, scrivere un algoritmo che stabilisce se la CC contenente s è un sottografo che, preso singolarmente, è un grafo completo.

comp. connessa ↑ CC



S G

* componente che contiene s.

VOGLIO SAPERE SE È COMPLETA (presa singolarmente).

Grafo completo: $\forall u \in V \ \forall v \in V \setminus \{u\} \ (u,v) \in E$ → per grafi orientati

$\forall u \in V \ \forall v \in V \ (u,v) \in E$ → per grafi non orientati.

→ un grafo completo contiene il numero massimale di archi.

↓

= "ogni nodo è connesso con tutti gli altri".

non orientati) n vertici = $\frac{(n-1)(n)}{2}$ archi effettivi

```
BFS(G,s):        G(V,E)   s ∈ V
for u ∈ V \ {s}
    u.col = "WHITE"
    u.d = ∞
    u.π = NIL
Enqueue (Q, s);
s.col = "gray"
s.d = 0 ;
s.π = ∅ ;

while Q ≠ ∅
    u = Dequeue (Q)
    for v ∈ Adj[u] {
        IF ( v.color == "white" ):
            v.color = "gray"
            v.π = u ;
            v.d = u.d + 1 ;
            Enqueue (Q,v)
    u.color = "Black" ;
}
```

vertici ++ →

archi ++ →

↓

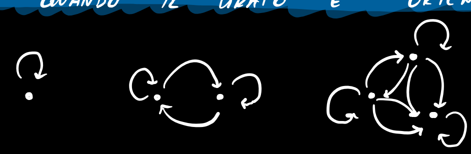
* Posso fare i miei controlli!

```
BFS_COMPLETE (G,s)
e = e/2 ;
IF e == n·(n-1)/2
    return TRUE ;
else
    return FALSE ;
```

→ verifica di prima.

Se orientato e completo, faccio questa VERIFICA →

E QUANDO IL GRAFO È ORIENTATO?



1 → 1 2 → 4 3 → 9

Insomma: n^2 ARCHI