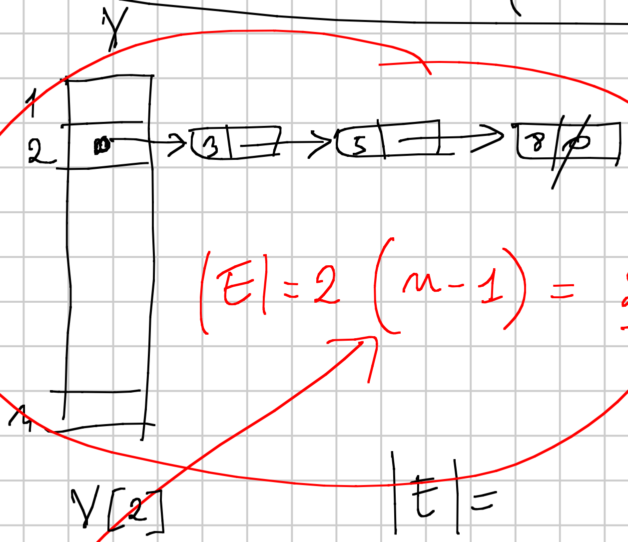
Dato un grafo $G=(V,E)$ non è orientato. Progettare un algoritmo efficiente per stabilire se G è un albero.
(liste di adiacenza)



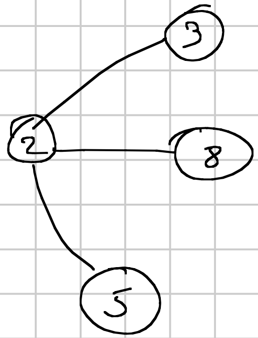
$V[2]$

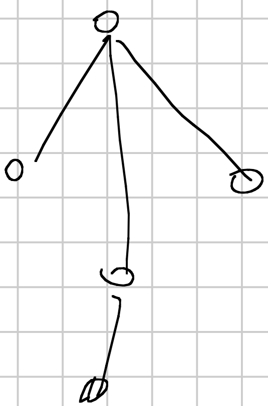
$|E| = 2(n-1) = 2n-2$

$|E| =$

u $\quad$ $Adj(u)$

$u=2$





G è un albero se è connesso e privo di cicli.

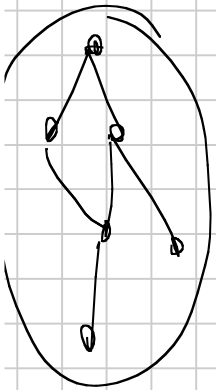
Es 1.

Titolo nota 17/03/2016

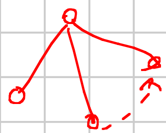
Albero? (G, 1)

tipo BFS

e = numero di archi



```
for ogni u ∈ V-{1} {
   u.colore = bianco }   e = ∅;
   Q = ∅;   ENQUEUE(Q, 1);
   while Q ≠ NIL {
   u = DEQUEUE(Q);
   for ogni v ∈ Adj(u) {
      if (v.colore == bianco) {
           v.colore = grigio;   e = e+1;
           ENQUEUE(Q, v)
      } else if (v.colore == grigio)   return false;
   } u.color = black; }
```



2n - 1

$\Theta(|V| + |E|)$

$\Theta(|V|)$

Albero? (G, 1)

```
if (e != 2n - 1) return false
(for ogni v ∈ V
   if (v.color == bianco) return false)
return true;
```

$G = (V, E)$ orientato e x, y, z 3 vertici $\in V$

Titolo nota 17/03/2016

Stabilire se y si trova nel cammino tra x e z.

DFS

Percorso (G, x, y, z)

for ogni $v \in V$ v.color = bianco;

$(G, 0, 6)$ → DFSpercorso (G, x, y);

if (y.colore == bianco) return false

→ for ogni $v \in V$ v.color = bianco;

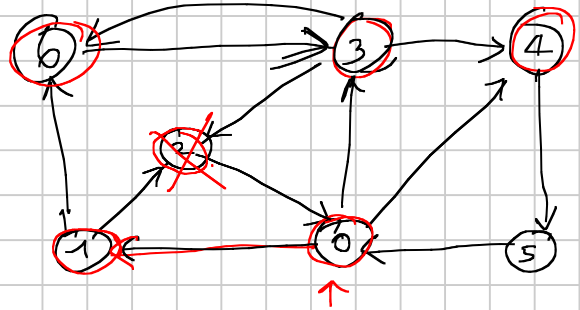
$(G, 6, 4)$ DFSpercorso (G, y, z);

if (z.color == Bianco) return false;

return true;

$\Theta(|V| + |E|)$

percorso$(G, 0, 6, 4)$



DFS percorso (G, u, d);

u.color = grigio;

for ogni $v \in Adj(u)$ {

if (v.color == bianco) {

if (v == d) {

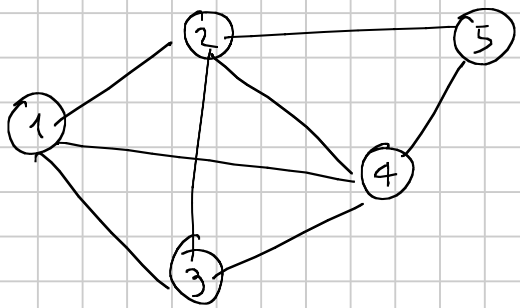
v.color = Grigio;

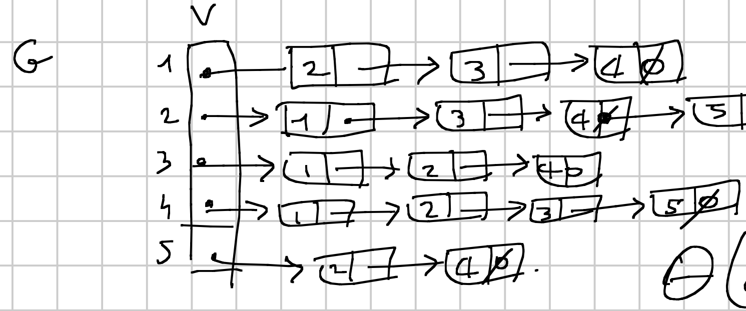
return; }

else DFS percorso(G, v, d

$G = (V, E)$ memorizzato con liste di adiacenza.

Trasformare G in G' che contiene le stesse informazioni ma memorizzato in matrice di adiacenza.





$\Theta(n+m) = O(n^2)$

ListaMatrice(G)

A = nuova matrice $n \times n$ // $n = |V|$ G'

for $i = 1$ to n
  for $j = 1$ to n $A[i,j] = 0$; $\Theta(|V|^2)$

for ogni v
  for ogni $u \in Adj(v)$ $A[v,u] = 1$; $\Theta(|V|+|E|)$
return A;

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

A

tempo $= \Theta(|V|^2) + \Theta(|V|+|E|)$

$= \Theta(n^2) + \Theta(n+m) =$

$= \Theta(n^2)$

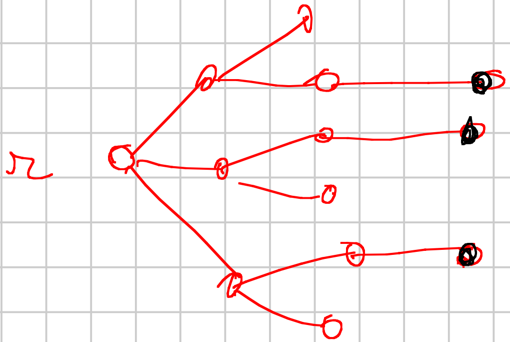
$n-1 \leq m \leq \frac{n(n-1)}{2} = \Theta($

CLIQUE di n nodi = grafo completo di n nodi $\left(\frac{n(n-1)}{2} \text{ archi} \right)$

<u>ES. 4</u> : $G(V,E)$ grafo connesso e non orientato.

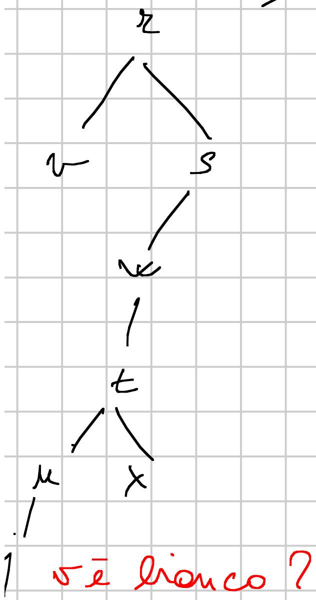
Alg che dato G e un vertice r, restituisce il numero di vertici che si trovano a distanza massima da r.

BFS



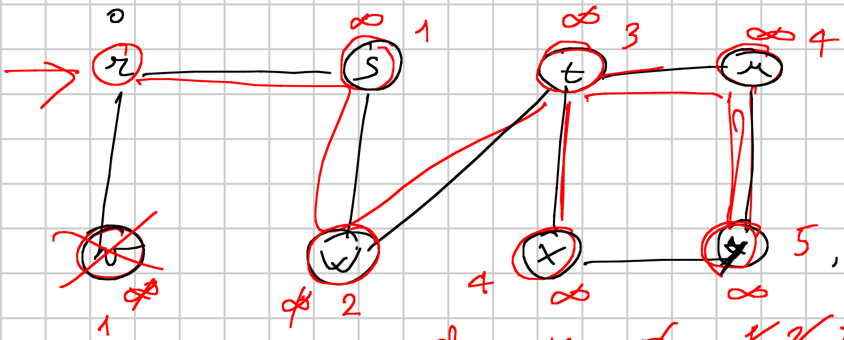
Distanzamax (G, r) da BFS

albero BFS



v è bianco?

```
dmax = 0;
ctr = 0;
for ogni v ∈ V - {r}   v.d = ∞;
r.d = 0;   Q = nuova coda;
Enqueue (Q, r);
while Q ≠ ∅ {
    u = DEQUEUE (Q);
    for ogni v ∈ Adj(u) {
        if (v.d == ∞) { v.d = u.d + 1;
            if (v.d > dmax) { dmax = v.d;  ctr = 1;}
            else if (v.d == dmax) ctr++;
        ENQUEUE (Q, v);
} } }
```



dmax = ~~0~~ = ~~1~~ ~~2~~ ~~3~~ 4

ctr = ~~0~~ ~~1~~ ~~2~~ ~~1~~ ~~1~~ ~~2~~ 1 5

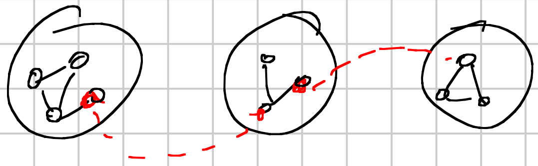
Q ~~r~~, ~~v~~, ~~s~~, ~~w~~, ~~t~~, ~~u~~, ~~x~~, ~~y~~

u 1 5

$\Theta(n+m)$

G (V, E) non orientato, progettare un algoritmo che restituisca il minimo <u>numero</u> di archi che si devono aggiungere a G per renderlo connesso.

G



= il n° di componenti connesse − 1

si modifica DFS

Soluzione = # minimo di archi = # componenti connesse − 1

Titolo nota 17/03/2016

Connetti(G)

numcc = 0;

for ogni $v \in V$ { v.color = bianco; v.π = NIL; v.d = v.f = 0; }

for ogni $v \in V$ {

if (v.color == bianco) { numcc++;

DFS-visit(G, v)   uso la procedura del libro

}

} return numcc-1;

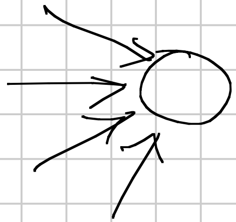
$\Theta(|V|+|E|)$

Pozzo in un grafo orientato è un vertice di grado uscente 0 e grado entrante $= n-1$

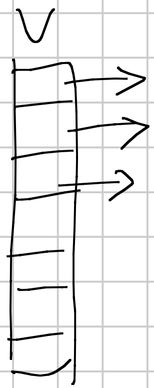
$\boxed{O(n \cdot m)}$

$\Theta(|V|+|E|)$

$n-1$ archi



dove $n = |V|$



Se esiste un pozzo questo è unico.

Scrivere un algoritmo per trovare un pozzo in G se esiste.

Si risolve con l'analisi delle liste di adiacenza

TrovaPozzo(G) Costruisce ge che contiene il grado entrante di tutti i nodi

ge = nuovo array di dim. n // per memorizzare grado entrante

$\Theta(n)$ for ogni $v \in V$ ge[v] = 0;

$\Theta(n+m)$ for ogni $v \in V$

  for ogni $u \in Adj(v)$ ge[u]++;

$\Theta(n)$ for ogni $v \in V$

  if (Adj(v) == NIL && ge[v] = n-1)

    return v;