Esercitazione 15 maggio

Friday, April 17, 2020 11:01 AM

Es. 1 $G=(V,E)$ non orientato

G è un albero?

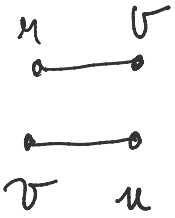
input: G memorizzato su liste di adiacenza

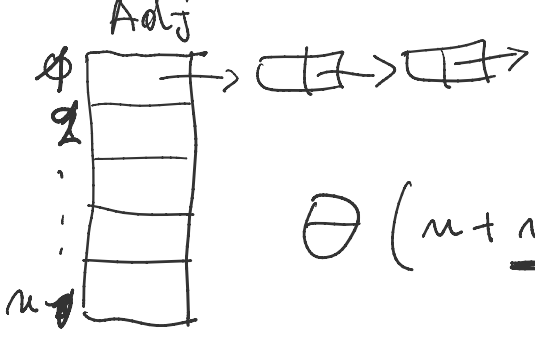
Albero:

1) Grafo non orientato, connesso e aciclico
2) Grafo non orientato, connesso e $|E| = |V| - 1$
3) " , aciclico e $|E| = |V| - 1$

Albero (G): $|V| = n$

```
e = 0;   // ctr degli archi
for all u ∈ V {
    for all v ∈ Adj[u] {
        e++;
        if (e > 2(n-1)) return FALSE;
    }
}
```





Scansione della Adj

$\Theta(n+m)$

if $(e < 2(n-1))$ return FALSE;

BFS $(G, 1)$; $\quad \Theta(n+m)$

for all $v \in V$ if $(v.color == white)$ return FALSE;

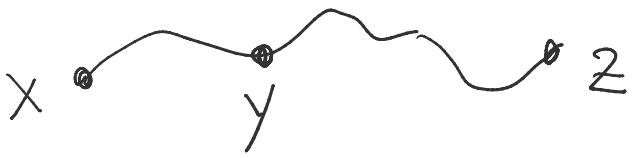
return True;

$m = \Theta(n)$ perché G è un albero

$\Theta(n)$

input $\begin{cases} G = (V, E) & \text{orientato} \\ x, y, z & \text{3 vertici di } G \end{cases}$

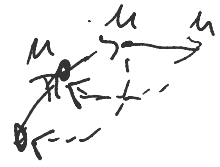
$\exists$ un cammino che unisce x a z e passa per y?



Definiamo un alg. (basato su DFS) che trova se $\exists$ un percorso tra 2 nodi.

DFS percorso (G, u, d):

$u.color = gray;$



u.color = gray;
for all v ∈ Adj[u] {
if (v.color == white) {
if (v == d) {
v.color = gray;
return;
}
else DFSpercorso(G, v, d)
}}

$\Theta(n+m)$

se d è bianco non c'è percorso

Percorso(G, x, y, z)
for all v ∈ V v.color = white;
DFS_Percorso(G, x, y);
if (y.color == white) return FALSE;
for all v ∈ V v.color = white
DFS_Percorso(G, y, z);
if (z.color == white) return FALSE;
else return TRUE;

uscita:

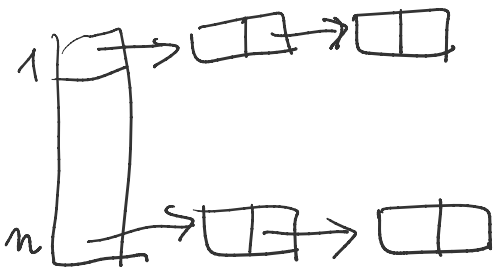
reinizz:

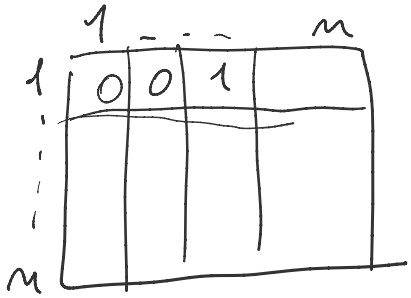
$\Theta(n+m)$

ES. 3 $G(V, E)$

$G \rightarrow G'$

$G \rightarrow G'$





lista matrice (G):

A = nuova matrice $n \times n$;

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

for all $v \in V$ {

$\Theta(n+m)$ for all $u \in Adj[v]$
    $A[v,u] = 1$;

}

return A;

$m = O(n^2)$

$\Downarrow$

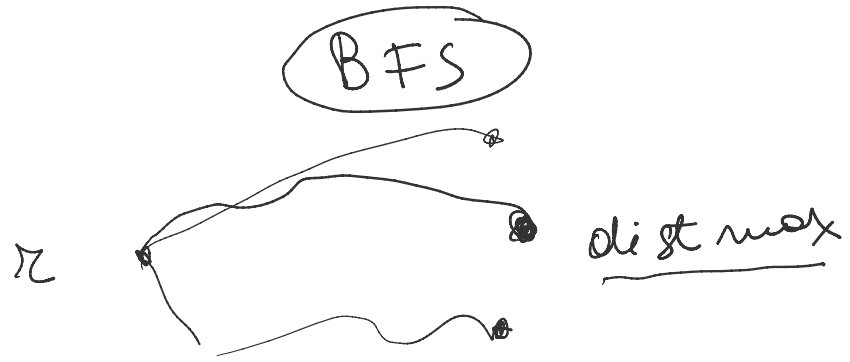
$\ominus(n^2)$

Es. 4

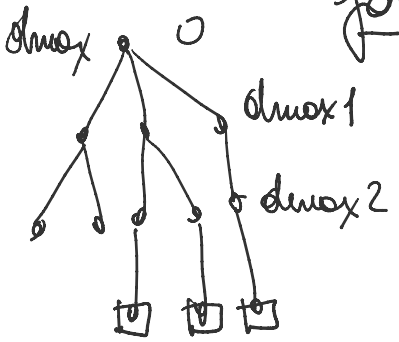
input: $G = (V, E)$ connesso e non orientato
r è la sorgente

output: n° di vertici che si trovano a
[illegible] distanza max da r

output: il valore della distanza max da r.




```
Distanza Max (G, r):
dmax = 0;
ctr = 0;
for all v ∈ V \ r    v.d = ∞;
r.d = 0;
Q = nuovaCoda ();
Enqueue (Q, r);
While (Q ≠ ∅) {
    v = Dequeue (Q);
    for all u ∈ Adj [v] {
        if (u.d == ∞) {
            u.d = v.d + 1;
            if (u.d > dmax) {
                dmax = u.d;
                ctr = 1;
            } else if (u.d == dmax)
```

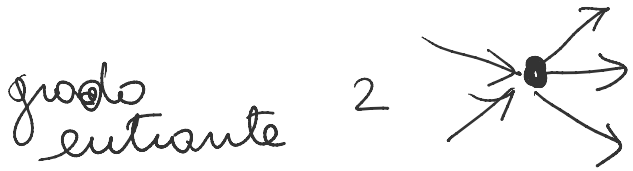


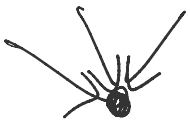
```
        } else if (u.d == dmax)
                                ctr++;
            Enqueue(Q, u);
        }
    }
} return ctr;
```

$\Theta(m+n) \equiv$ BFS

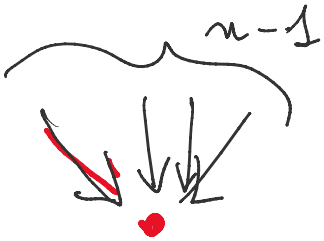
Es. 5 G grafo orientato

pozzo = vertice a grado uscente = 0
" a grado entrante $n-1$





POZZO SORGENTE

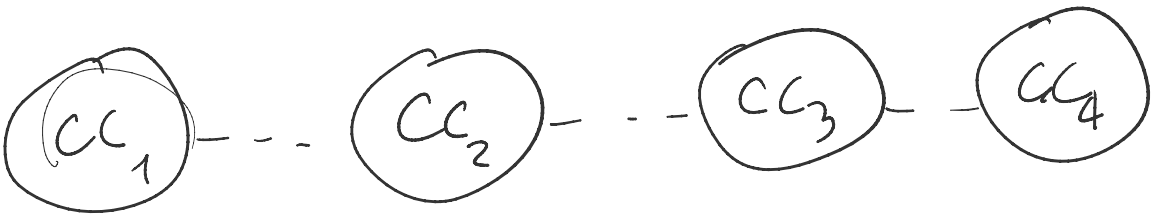


grado entrante $= n-1$
grado uscente $= 0$

Stabilire se G ha un pozzo.

Es 6 Dato $G(V,E)$ non orientato progettare un alg. che restituisca il numero minimo di archi da aggiungere a G per renderlo connesso

n° = n° Componenti connesse − 1



Componenti connesse

DFS

```
Connetti(G):                    use DFS visit
numCC = 0;
for all v ∈ V { v.colore = white }
for all v ∈ V {
    if (v.color == white) {
        numCC++;
        DFS-visit(G, v)
    }
}
```

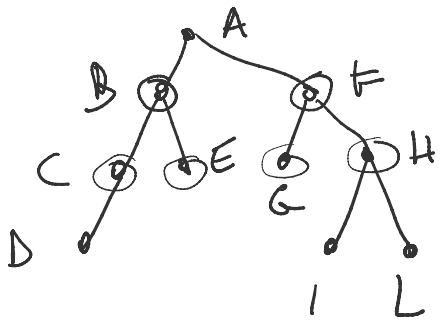
$\Theta(n+m)$

}

return num_CC - 1;

Dato un albero binario T lo vogliamo visitare per livelli

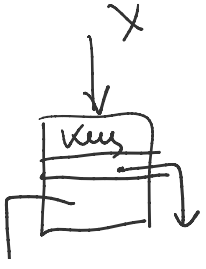
Visita anticipata - DFS
Visita simmetrica
Visita differita

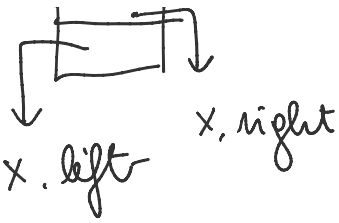


⇒ A B F C E G H D I L

Struttura dati ⇒ coda semplice
FIFO

```
VisitaLivelli (T)
if (T.root ≠ NIL) {
    Q = nuova coda ();
    Enqueue (T.root, Q);
    while (Q ≠ ∅) {
        x = Dequeue (Q);
```





$\Theta(n)$

```
x = Dequeue(Q);
print x.key;
if (x.left != NIL)
    Enqueue(Q, x.left);
if (x.right != NIL)
    Enqueue(Q, x.right);
}
}
```