ESERCITAZIONE sui GRAFI

Titolo nota 26/02/2016

① Algoritmo efficiente per stabilire se un grafo è un albero.

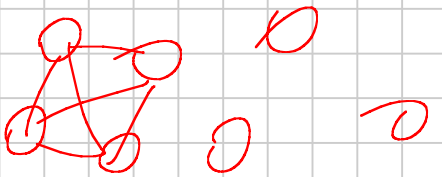
Albero: Grafo non orientato, connesso e aciclico

$\Leftrightarrow$

$\rightarrow$ Grafo non orientato, connesso e $|E| = |V| - 1$

$\Leftrightarrow$

Grafo non orientato, aciclico e $|E| = |V| - 1$



obiettivo: algoritmo di complessità

$$T(|V|, |E|) = O(|V|)$$

idea: conto gli archi e poi verifico se è connesso con una visita (si fa solo se $|E| = |V| - 1$)

```
Albero(G)  // n = |V|
e = 0 // ctr doppi archi
for all u ∈ V {
    for all v ∈ Adj(u) {
        e++;
        if (e > 2(n-1)) return FALSE  // ci sono troppi archi → il grafo non è aciclico
    }
}
```

- `if (e < 2(n-1)) return FALSE` // gli archi sono troppo pochi, il grafo non è connesso

```
// verifica che G sia connesso:
BFS(G, 1)  // 1 = primo vertice nella rappresentazione di G.
for all v ∈ V {
    if (v.color == B) return FALSE;  // il grafo non è connesso
}
return TRUE;
```

Analisi

$$T(|V|, |E|) = \Theta(|V|) + O(|V| + |E|) = \Theta(|V|) + O(|V|) = \Theta(|V|)$$

- $\Theta(|V|)$: costo del calcolo del # archi
- $O(|V| + |E|)$: costo della visita. $|E| = |V| - 1$

(2) $G = (V, E)$ grafo orientato.

$x, y, z \in V$

Stabilire se y si trova su un cammino orientato $x \rightsquigarrow z$

$$\exists\ x \overset{y}{\rightsquigarrow} z$$

$$\Leftrightarrow$$

$$\exists\ x \rightsquigarrow y \ \&\ \exists\ y \rightsquigarrow z$$

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

```
DFSpercorso(G, u, d)
u.color = GRIGIO
for all v ∈ Adj(u) {
    if (v.color == BIANCO) {
        if (v == d) {
            v.color = GRIGIO;
            return;
        }
        else DFSpercorso(G, v, d)
    }
}
```

ANALISI: $T(|V|, |E|) = O(|V| + |E|)$ si eseguono al massimo due visite

ESERCIZIO 3

Titolo nota 26/02/2016

G: grafo memorizzato con liste di adiacenza

Costruire G', G' = G ma memorizzato con matrice di Adiacenza.

Lista2Matrice(G)

```
A = nuova matrice n×n // n = |V|
for i = 1 to n {
    for j = 1 to n {
        A[i,j] = 0;
    }
}
for all v ∈ V {
    for all u ∈ Adj[v] {
        A[v,u] = 1;
    }
}
return A;
```

Analisi:

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

ESERCIZIO 4

Titolo nota 26/02/2016

Dato G connesso e non orientato, e dato $r \in V$, progettare un algoritmo per calcolare il # di vertici a distanza massima da r.

↓

BFS

```
DistanzaMax(G, r)
dmax = 0;
ctr = 0;
for all v ∈ V[G] { v.d = ∞ };
r.d = 0;
Q = nuova Coda();
Enqueue(Q, r);
while (Q ≠ ∅) {
    v = Dequeue(Q);
```

```
for all u ∈ Adj(v) {
    if (u.d == ∞) { // u è raggiunto per la prima volta
        u.d = v.d + 1;
        if(u.d > dmax) { dmax = u.d; ctr = 1 }
        else if (u.d == dmax) ctr++;
        Enqueue(Q, u);
    }
}
return ctr;
```

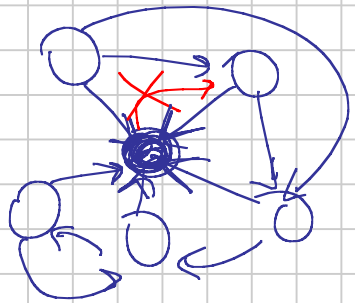
$T(|V|, |E|) = \Theta(|V| + |E|)$

(G connesso)

ESERCIZIO 5

Titolo nota 26/02/2016

Un pozzo in un grafo orientato G è un vertice di grado entrante $n-1$ e grado uscente 0:
Se il pozzo esiste, è unico.
Scrivere una procedura in pseudocodice per trovare il pozzo in G, se esiste.



Soluzione con calcolo del "grado entrante"

```
TrovaPozzo(G)   // n = |V|
  ge = nuovo array di dim. n   // per memorizzare il grado entrante
  for all v ∈ V { ge(v) = 0; }
  for all v ∈ V {
      for all u ∈ Adj[v] {        // arco (v,u)   v → u
          ge[u]++;
      }
```

```
}
for all v ∈ V {
    if (qe[v] == n-1 && adj(v) == NIL)   return v;
                 //
}
return NIL;
```

$\delta_e(v) = n-1$ $\qquad$ $\delta_u(v) = 0$

Analisi: $T(|V|, |E|) = \Theta(|V| + |E|)$

$S(|V|, |E|) = \Theta(|V|)$

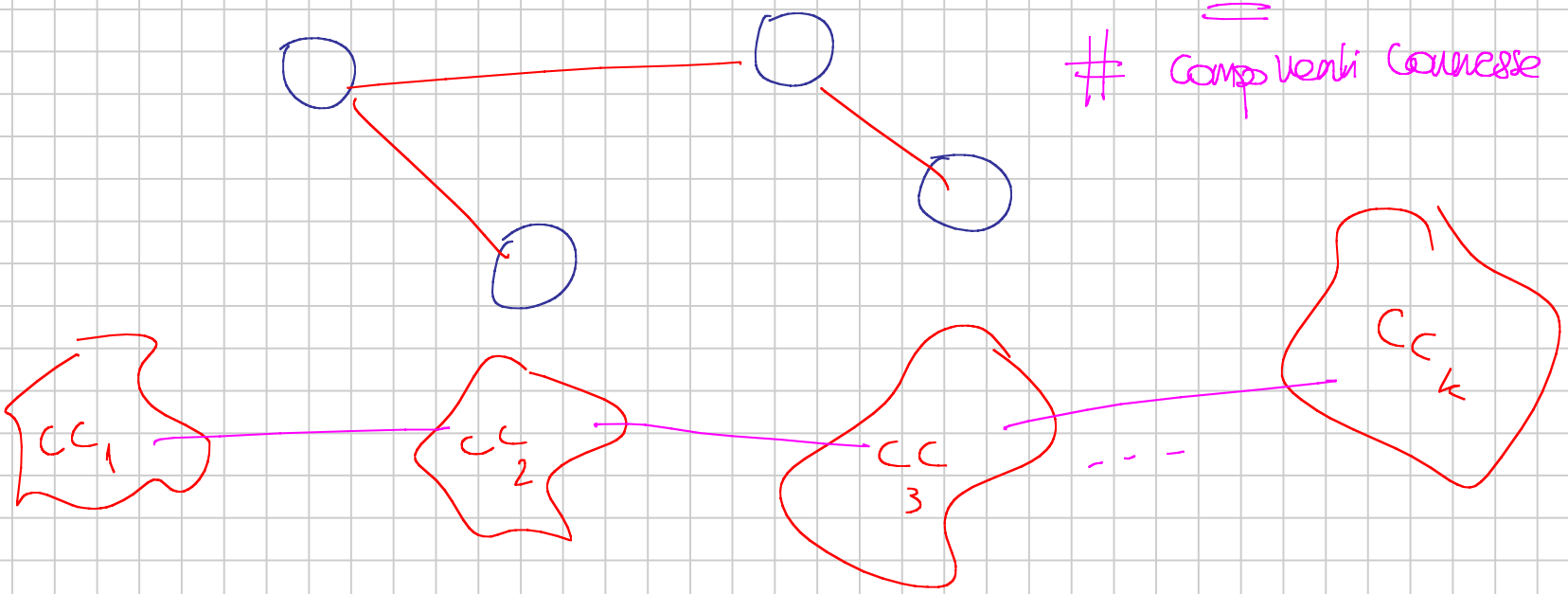
array qe

ESERCIZIO (6)

Titolo nota 26/02/2016

Dato $G=(V,E)$ non orientato, progettare un algoritmo che restituisca il MINIMO numero di archi da aggiungere a G per renderlo connesso.

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



```
Connetti(G)
numCC = 0;
for all v∈V { v.colore = BIANCO }
for all v∈V {
    if(v.colore == B) {
        numCC++;
        DFS-visit(G, v);
    }
}
return numCC - 1;
```

Analisi:

$T(|V|, |E|) = \Theta(|V| + |E|)$

$S(|V|, |E|) = O(n)$

× la gestione delle chiamate ricorsive

Titolo nota 26/02/2016

ESERCIZIO — Visita per livelli di un albero binario

↳ usiamo **BFS**

↓ memorizzato con puntatori .left e .right

```
VisitaLivelli(T)
if (T.root ≠ NIL) {
    Q = nuovaCoda();
    Enqueue(T.root);
    while (Q ≠ ∅) {
        x = Dequeue(Q);
        print x.key;
        if (x.left ≠ NIL) Enqueue(Q, x.left);
        if (x.right ≠ NIL) Enqueue(Q, x.right);
    }
}
```

ESERCIZIO

Titolo nota 26/02/2016

Dato $G=(V,E)$ non orientato, e dati $x, y, z \in V$ progettare un algoritmo che restituisca:

1. se x, y, z $\in$ alla stessa componente connessa
2. " " $\in$ a due componenti connesse $\neq$
3. " " $\in$ a tre C.C. diverse.

```
CCC(G, x, y, z)
for all v ∈ V { v.color = B };
BFS(G, x);
if ( y.color ≠ B && z.color ≠ B ) return 1;   // x, y, z nella stessa CC
if ((y.color == B && z.color ≠ B) || (y.color ≠ B && z.color == B))
                return 2;
for all v ∈ V { v.color = B }                 // non è indispensabile in questo caso
```

```
BFS(G, y);
if (z.color ≠ B) return 2;
else return 3;
```

ESERCIZIO

Dato un grafo orientato $G = (V, E)$, memorizzato con le liste di adiacenza, e un intero $k < |V|$, stabilire se G ha almeno k vertici che hanno lo stesso grado entrante.