ESERCIZIO 1

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

Soluzione

G è un albero

$\iff$

G è connesso e $|E| = |V| - 1$

$\Rightarrow$ si contano gli archi.
se il # di archi ~~[illegible]~~ raggiunge il valore $n=|V|$ si restituisce FALSE.

Se $|E| < |V| - 1$ si restituisce FALSE (in questo caso G non è connesso)

Se $|E| = |V| - 1$ si verifica se G è connesso con una visita BFS.

La visita si esegue solo se $|E| = |V| - 1$

$\Rightarrow$ il costo sarà

$$T(n, m) = O(n+m) = O(n+n-1) = O(n)$$

dove $n = |V|$, $m = |E|$.

Albero (G)

```
e=0;
for(u=0; u<n; u++) {
    for(x=Adj[u].inizio; x≠null; x=x.next) {
        e++;
        if (e>2(n-1)) return FALSE;
                        //ogni arco si conta 2 volte
    }
}
if (e<2(n-1)) return FALSE;  // G non è connesso
BFS(G,0);   ~> O(|V|+|E|) = O(n+n-1) = O(n)
for (u=0; u<n; u++) {
    if(!raggiunto[u]) return FALSE;
}
return TRUE;
```

$$T(n, m) = O(n)$$

ESERCIZIO 2

Sia $G = (V, E)$ un grafo orientato.
Siano $x, y, z \in V$.
Stabilire se y si trova su un cammino $x \rightsquigarrow z$.

Suggerimento

y è su un cammino $x \rightsquigarrow z$

$\Longleftrightarrow$

$\exists$ un cammino $x \rightsquigarrow y$ e un cammino $y \rightsquigarrow z$

Soluzione

Percorso(G, x, y, z)

```
for (u=0; u<n; u++) raggiunto(u)= false;
DFSpercorso (x, y);
if (raggiunto [y] == false) return FALSE;
for (u=0; u<n; u++) raggiunto (u) = false;
DFSpercorso (y, z);
return raggiunto (z);
```

DFSpercorso (u,d) // u e d sono vertici

```
raggiunto (u) = true;
for (x= adj (u). inizio; x ≠ null; x=x.next) {
    v= x.dato;
    if (! raggiunto (v)) {
        if (v==d) {
            raggiunto[v] = true;
            return;
        }
        else DFSpercorso (v, d);
    }
}
```

$$T(n, m) = O(n + m)$$

Si eseguono al massimo 2 visite del grafo G.

ESERCIZIO 3

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

Progettare un algoritmo che trasformi G nel grafo G' che contiene la stessa informazione ma è memorizzato con una matrice di adiacenza.

Matrice (G)

$\Theta(n^2)$

```
A = nuova matrice n×n      // n = |V|
for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
        A(i,j) = 0;
    }
}
```

$\Theta(n+m)$

```
for (u=0; u<n; u++) {
    for (x = adj(u).inizio; x ≠ null; x = x.next) {
        v = x.dato
        A(u,v) = 1;
    }
}
return A;
```

$n = |V|$

$m = |E|$

$m = O(n^2)$

$$\boxed{T(n,m) = \Theta(n^2)}$$

ESERCIZIO 4

Sia $G=(V,E)$ un grafo connesso e non orientato.

Progettare un algoritmo che ricevuto in ingresso G e un suo vertice r, restituisca il numero di vertici ~~[illegible]~~ che si trovano a distanza massima da r.

Soluzione

il problema si risolve eseguendo una BFS con sorgente in r, e calcolando le distanze di ogni vertice da r.

DistanzaMax(G, r)

```
dmax = 0;
ctr = 0;
for (u = 0; u < n; u++) { distanza(u) = +∞; }
distanza(r) = 0;
Q = nuova Coda();
Q.enqueue(r);
while (Q non è vuota) {

    u = Q.dequeue();
    for (x = Adj(u).inizio; x ≠ null; x = x.next) {

        v = x.dato;
        if (distanza(v) == +∞) { // v non è ancora stato raggiunto
            distanza(v) = distanza(u) + 1;
            if (distanza(v) > dmax) {
                dmax = distanza(v);
                ctr = 1;
            }
            else if (distanza(v) == dmax) {
                ctr++;
            }
            Q.enqueue(v);
        }
    }
}
return ctr;
```

$T(n, m) = O(n+m)$

ESERCIZIO 5

Dato un grafo orientato G, un pozzo è un vertice con grado uscente 0 e grado entrante uguale a $|V| - 1$.
Si osservi che, se esiste, il pozzo è unico.
Scrivere una procedura in pseudocodice per trovare il pozzo in G, se esiste.

Soluzione

occorre calcolare il grado entrante dei nodi

TrovaPozzo(G)

```
~~for all~~
for (u = 0; u < n; u++) { ge(u) = 0; }                      Θ(|V|)
for (u = 0; u < n; u++) {
    for (x = Adj(u).inizio; x ≠ null; x = x.next) {
        v = x.dato;
        ge(v)++;
    }
}                                                           Θ(|V|+|E|)
for (u = 0; u < n; u++) {
    if (Adj(u).inizio ≠ null && ge(u) == n-1)
        return u;
}                                                           O(|V|)
return null;
```

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

ESERCIZIO 6

Dato un grafo non orientato, progettare un algoritmo che restituisca il minimo numero di archi da aggiungere al grafo per ~~rendl~~ renderlo connesso.

Soluzione

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

Connetti (G)

```
numCC = 0;
for (u=0; u<n; u++) raggiunto(u) = false;
for (u=0; u<n; u++) {
    if (!raggiunto(u)) {
        numCC++;
        DFSric(u);
    }
}
return numCC - 1;
```

DFSric(v): vedi libro di testo.

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