**Esercizio 1.1** (Nodi gemelli). *Dato un grafo $G$ tale che $|V(G)| \geq 2$, dimostrare che esistono almeno due nodi distinti con lo stesso grado*

Supponiamo che ogni nodo ha grado diverso, ordiniamo tali nodi nel modo seguente: $\{v_1, v_2 \dots, v_n\}$ con $i > j \Rightarrow deg(v_i) > deg(v_j)$. Il nodo $v_1$ ha $k \geq 1$ adiacenti, il nodo $v_2$ ha $k' > k \geq 1$ adiacenti, e' chiaro che il nodo $v_n$ avra' ALMENO $n$ adiacenti, ma ci sono al piu' $n-1$ nodi che possono esserygli adiacenti $\Rightarrow$ CONTRADDIZIONE.

**Esercizio 1.2** (Grafo complementare). *Dato un grafo $G$, definiamo $\overline{G}$ come grafo complementare di $G$ se per ogni arco $(u, v)$ si verifica che $(u, v) \in E(G)$ se e solo se $(u, v) \notin E(\overline{G})$. Dimostrare che almeno uno tra $G$ e $\overline{G}$ è connesso.*

Supponiamo che entrambi i grafi non siano connessi

**Esercizio 1.3** (Grado minimo $\frac{n}{2}$). *Dimostrare che un grafo non diretto $G$ in cui ogni nodo ha grado maggiore o uguale a $\left\lceil \frac{|V(G)|}{2} \right\rceil$ è sempre connesso. L'affermazione vale anche se $G$ è diretto?*

**Esercizio 1.4** (Arcipelago). *Un arcipelago è rappresentato da una matrice $n \times m$, dove ogni cella è marcata da uno 0, rappresentante il mare, o da un 1, rappresentante il terreno. In particolare, due celle appartengono alla stessa isola se e solo se sono marcate entrambe con 1 e sono adiacenti. Data in input la matrice $M$, progettare un algoritmo che in tempo $O(nm)$ restituisca il numero di isole nell'arcipelago. Modificare l'algoritmo affinché restituisca la dimensione dell'isola di grandezza maggiore.*

```
Arcipelago( M[n,m]: matrice) {
    c=1
    For(i=0...,n){ Θ(n)
        For(j=0...,m){ Θ(m)
            if(M[i,j]==1){
                K= adj_island(M,i,j)//Θ(9)
                if(K==1){
                    c++
                    M[i,j]=c
                }else{ M[i,j]=K }
            }
        }
    }
    ni=0
    For(i=0...,n){
        For(j=0...,m){
            ni=max(ni,M[i,j]
        }
    }
    return ni-1
}

Biggest_island(M[n,m]: matrice) {
    ni=Arcipelago(M)// la funzione modifica la matrice
    A[ni+1]={0,0...,0}
    For(i=0...,n-1){
        For(j=0...,m-1){
            if(M[i,j]≠0){ A[M[i,j]]++ }
        }
    }
    bi=0
    in:intero
    For(i=0...,ni+1){
        if(A[i]>bi){
            bi=A[i]
            in=i
        }
    }
    return in
}
```

```
adj_island( M[n,m]:matrice, x:intero, y:intero){
    K=1
    For(i=x-1...,x+1){ Θ(3)
        For(j=y-1...,y+1){ Θ(3)
            i=clamp(0,n-1)
            j=clamp(0,m-1)
            K=max(K,M[i,j])
        }
    }
    return K
}
```

**Esercizio 1.5** (Corsi propedeutici). *L'Università della Croce di Santa Minerva sta introducendo un nuovo corso di studi in Icosidodecaedrologia. Per gli $n$ esami di tale corso sono previste $m$ propedeuticità, ossia per sostenere alcuni corsi è necessario averne sostenuti prima altri. Ogni propedeuticità è rappresentata da una coppia $(e_i, e_j)$ se per sostenere l'esame $e_j$ è necessario sostenere prima l'esame $e_i$.*

*Progettare un algoritmo che dati in input il valore $n$ e un insieme $P$ contenente le $m$ coppie restituisca una possibile programmazione degli esami in modo che tutti possano essere sostenuti. Discutere il costo dell'algoritmo dato.*

Si può modellare un grafo con nodi $e_k$ ed archi $(e_i, e_j)$. Basta fare un ordinamento topologico.

```
Prop( P={(e_i,e_j)} ){
    G:grafo
    For each (e_i,e_j)∈P {
        V(G).add(e_i)
        V(G).add(e_j)
        E(G).add((e_i,e_j))
    }
    L:list
    Vis[n]={0,...,0}
    return Ord(G,L,Vis)
}
```

```
Ord(G:grafo, L:list, Vis:array) {
    For each x∈V(G) {
        iF(Vis[x]≠1){ DFS_rec(G,x,L,Vis) }
    }
    return L
}

DFS_rec(G:grafo, x:nodo, L:lista, Vis:array) {
    Vis[x]=1
    For each y~x {
        iF(Vis[y]≠1){ DFS_rec(G,y,L,Vis) }
    }
    L.add(x)
}
```

**Esercizio 1.8** (Cammino crescente più lungo). *Data una matrice $M$ di dimensioni $n \times m$ di numeri interi positivi, definiamo come cammino della matrice una serie di indici $(i_1, j_1), \ldots, (i_k, j_k)$ ottenuti spostandoci ad ogni passo solo verso le entrate verticalmente o orizzontalmente adiacenti. Inoltre, definiamo un cammino della matrice come crescente se $m_{i_1,j_1} < \ldots < m_{i_k,j_k}$.*

*Progettare un algoritmo di complessità $O(nm)$ che data in input la matrice $M$ restituisca la lunghezza il cammino crescente più lungo al suo interno.*

Creo un grafo diretto in cui ogni entrata della matrice e' un nodo ed ha un arco verso i nodi a destra/giu' di valore maggiore

```
Es8(M:matrice n×m){
    G:grafo
    For(i=1...,n){
        For(j=1...,m){
            V(G).add(i,j))
        }
    }
    For(i=1...,n){
        For(j=1...,m){
            iF(M[i,j]<M[i+1,j]){
                E(G).add((i,j),(i+1,j))
            }
            iF(M[i,j]<M[i,j+1]){
                E(G).add((i,j),(i,j+1))
            }
        }
    }
    return max_dist(G)//distanza massima fra 2 nodi
}
```