

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 essergli adiacenti \Rightarrow CONTRADDIZIONE.

Esercizio 1.2 (Grafo complementare). Dato un grafo G , definiamo \bar{G} come grafo complementare di G se per ogni arco (u, v) si verifica che $(u, v) \in E(G)$ se e solo se $(u, v) \notin E(\bar{G})$. Dimostrare che almeno uno tra G e \bar{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){  $\Theta(n)$ 
        For(j=0...,m){  $\Theta(m)$ 
            if(M[i,j]==1){
                K = adj_island(M,i,j) //  $\Theta(9)$ 
                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
}

```

```

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

```

```

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
}

```

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  $(e_i, e_j) \in P$  {  
         $V(G).add(e_i)$   
         $V(G).add(e_j)$   
         $E(G).add((e_i, e_j))$   
    }  
     $L$ : list  
     $Vis[n] = \{0, \dots, 0\}$   
    return Ord( $G, L, Vis$ )  
}
```

```
Ord( $G$ : grafo,  $L$ : list,  $Vis$ : array) {  
    For each  $x \in V(G)$  {  
        if ( $Vis[x] \neq 1$ ) { DFS.rec( $G, x, L, Vis$ ) }  
    }  
    return  $L$   
}  
  
DFS.rec( $G$ : grafo,  $x$ : nodo,  $L$ : lista,  $Vis$ : array) {  
     $Vis[x] = 1$   
    For each  $y \in x$  {  
        if ( $Vis[y] \neq 1$ ) { 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 $(i_1, j_1), \dots, (i_k, j_k)$ ottenuti spostandoci ad ogni passo solo verso le entrate verticalmente o orizzontalmente adiacenti. Inoltre, definiamo un cammino della matrice come crescente se $m_{i_1, j_1} < \dots < m_{i_k, j_k}$.

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 è un nodo ed ha un arco verso i nodi a destra/giù di valore maggiore

```

Es8(M:matrice nxm){
    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 maxDist(G) // distanza massima fra 2 nodi
}

```

Esercizio 2.2 (Voli di costo minimo). Un'azienda sta organizzando un evento in due sedi: una situata a Milano ed una a Roma. Per entrambe le sedi, l'azienda deve inviare n impiegati situati in città sparse. Per ognuno dei $2n$ impiegati viene calcolato il costo necessario per il viaggio verso Milano e il viaggio verso Roma. Tali costi sono descritti da una coppia (m_i, r_i) , dove m_i e r_i sono rispettivamente il costo del viaggio verso Milano e verso Roma dell' i -esimo impiegato.

Progettare un algoritmo di costo $O(n \log n)$ che date in input le $2n$ coppie $(m_1, r_1), \dots, (m_{2n}, r_{2n})$ restituisca i due insiemi di n impiegati che minimizzano la spesa dell'azienda.

```

Voli ( R : { (m1, r1) , ..., (m2n, r2n) } ) {
    M = R.copy()
    R = ordina secondo ri
    M = ordina secondo mi
    c = 0
    while (R ≠ ∅ ∧ M ≠ ∅) {
        r = R[0]
        R.remove(r)
        m = M[0]
        M.remove(m)
        c += m[0] + r[1]
    }
    return c
}

```

NOTAZIONE

$k = (m_i, r_i) \Rightarrow \begin{cases} k[0] = m_i \\ k[1] = r_i \end{cases}$

Esercizio 2.3 (Shopping di lusso). Siria ha recentemente fatto shopping sfrenato per articoli di lusso in una boutique esclusiva, all'interno della quale vi erano n articoli da cui scegliere. Siria, essendo nota per la sua ampia ricchezza, ha utilizzato una "parsimoniosa" strategia di acquisto: ispezionando ogni articolo dal primo all'ultimo, ha acquistato l'articolo attualmente ispezionato solo se il suo prezzo era inferiore ai fondi ancora disponibili, ignorandolo e passando al successivo se tali fondi fossero insufficienti per acquistarlo.