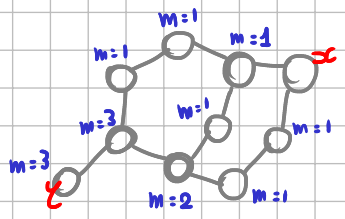
Dato un grafo $G = (V, E)$ non diretto e $x, y \in V$ scrivere lo pseudocodice di un algoritmo che conta il numero di cammini minimi da x a y in $O(|V| + |E|)$

Se la distanza da x a y è K, conto i nodi adiacenti a y che distano $K-1$ da x. Così a ritroso in maniera ricorsiva.

m: "modi per arrivare qui da x"




```
Min_paths_glob(G:grafo, x : nodo, y : nodo){
    Dist[n]={-1,-1...,-1}
    BFS(G, x, Dist) // riempio il vettore distanze
    return min_path(y, Dist)
}

min_path(y: nodo, Dist: array){
    if( |{ω | ω~y ∧ Dist[ω]=Dist[y]-1}|==0){ // caso base
        return 1
    }
    sum = 0
    for each(ω~y){
        if(Dist[ω]=Dist[y]-1){
            sum += min_path(ω, Dist)
        }
    }
    return sum
}
```

Dato un grafo $G = (V, E)$ con la proprietà che $\deg(v) \geq 2\ \forall v \in V$ (ogni vertice ha grado ≥ 2), scrivere lo pseudocodice di un algoritmo che trova un ciclo nel grafo in $O(|V| + |E|)$

```
DFS_ric(G:grafo, x:nodo, Vis : array, Ter: array){
    Vis[x]=1
    Ter[x]=-1
    for each(y~x){
        if(Ter[y]==0){
            DFS_ric(G,y,Vis,Ter)
        }
        else if(Ter[y]==-1){ // sto valutando un nodo visitato ma non chiuso
            L:set
            for(i=0...,n-1){
                if(Ter[i]==-1){ L.add(i)}
            }
            return L
        }
    }
    Ter[x]=1
}

Glob(G:grafo){
    x = V(G)[0]
    Dist[n] : array inizz. a 0
    Ter[n] : array inizz. a 0
    return DFS_ric(G, x, Vis, Ter)
}
```

Dato un grafo $G = (V, E)$ e due vertici $x, y \in V$ scrivere lo pseudocodice di un algoritmo che determina se esiste un cammino da x a y in $O(|V| + |E|)$

```
Find_path(G:grafo, x:nodo, y:nodo){
    bool t:False
    Vis = calloc(n)
    DFS(G, x, y, t, Vis)
    return t
}

DFS(G:grafo, x:nodo, y:nodo, t:bool, Vis: array){
    if(x == y){
        t=true
        return
    }
    Vis[x] = 1
    For each(w v x){
        if(Vis[w]==0){
            DFS(G, w, y, t, Vis)
        }
    }
}
```

Dato un grafo non diretto $G = (V, E)$ scrivere lo pseudocodice di un algoritmo che trova le **componenti connesse** del grafo in $O(|V| + |E|)$

- L'output è un array `a` di lunghezza $|V|$ in cui `a[v] = i` indica che il vertice `v` sta nell' `i`-esima componente connessa

```
DFS_comp(G:grafo, x:nodo, Vis: array, c:intero, comp:array){
    Vis[x] = 1
    comp[x] = c
    For each(y v x){
        if(Vis[y]==0){
            DFS(G, y, Vis, c, comp)
        }
    }
}

Comp(G:grafo){
    x = V(G)[0]
    Vis = calloc(n)
    Comp = calloc(n)
    For(i=1...n){
        if(comp[i]==0){
            DFS_comp(G, i-1, Vis, i, comp)
        }
    }
    return comp
}
```

Dato un grafo **diretto** $G=(V,E)$ scrivere lo pseudocodice di un algoritmo che trova gli archi **in avanti**, all'**indietro** e di **attraversamento** in $O(|V|+|E|)$

```
DFS_cc(G:grafo, x:nodo, t:array, T:array, cc:Intero, A:set){
    cc++
    t[x]=cc
    for each(y∈x){
        if(t[y]==-1){
            DFS_cc(G, y, t, T, cc, A)
            A.add((x,y)) // arborescenza
        }
    }
    T[x]=cc // la variabile cc e' passata per riferimento e si aggiorna dinamicamente
}

Archi(G:grafo, x:nodo){
    A:set
    t: array lungo n inizz. a -1
    T: array lungo n inizz. a -1
    cc=0
    DFS_cc(G, x, t, T, cc, A)
    E=E(G)\A // archi non dell' arborescenza
    alt:Set
    ind:Set
    avz:Set
    for each (x,y)∈E {
        if( [t[x],T[x]] ⊆ [t[y],T[y]]){
            ind.add((x,y))
        } if( [t[x],T[x]] ⊇ [t[y],T[y]]){
            avz.add((x,y))
        } if( [t[x],T[x]] ∩ [t[y],T[y]] == ∅){
            alt.add((x,y))
        }
    }
    return avz, ind, alt
}
```

Dato un grafo diretto $G = (V, E)$ un **pozzo universale** è un vertice $x \in V$ tale che

- $\nexists y \in V \mid (x, y) \in E$ (non ha archi uscenti)
- $\forall y \in V : y \neq x \implies (y, x) \in E$ (ogni altro vertice y ha un arco che entra in x)

Dato G come **matrice di adiacenza** scrivere lo pseudocodice di un algoritmo che verifica se G ha un pozzo universale in $O(|V|)$

```
Pozzo(G:grafo){
    S = V(G)
    i = 0
    For(J=1..., n-1){
        iF( M[i][J]==1){ // (i)→(J) => i non e' pozzo
            S.remove(i)
            i++
        }
        if(M[i][J]==0 ∧ i≠J){ (i)-X-▸(J) => J non e' pozzo
            S.remove(J)
        }
    } // a questo punto in S ho il candidato ad esseve pozzo
    if(S≠∅){
        x=S[0]
        if(x e' pozzo){ // controllo in O(n)
            return x
        }
    }
    return NULL
}
```

Dato un albero rappresentato come **vettore di padri** scrivere lo pseudocodice di un algoritmo che trova gli antenati di un vertice v in $O(n)$

```
Antenati(P:array, v:nodo){
    A:Set
    while(P[v]≠v){
        A.add(P[v])
        v=P[v]
    }
    return A
}
```

Dato un albero rappresentato come vettore di padri scrivere lo pseudocodice di un algoritmo che trova gli antenati comuni di due vertici v e w in $O(n)$

```
Antenati_comuni(P:array, x:nodo, y:nodo){
    AX: array lungo n inia. a 0
    AY: array lungo n inia. a 0
    while(x≠P[x]){
        Ax[P[x]]=1
        x=P[x]
    }
    while(y≠P[y]){
        AY[P[y]]=1
        y=P[y]
    }
    A: array lungo n inia. a 0
    For(i=0..., n-1){
        A[i]:AX[i] ∧ AY[i]
    }
    return A
}
```

Dato un albero rappresentato come vettore di padri scrivere lo pseudocodice di un algoritmo che trova il primo antenato in comune di due vertici v e w in $O(n)$

```
FCA ( P:vettore, v:nodo, w:nodo){
    AV = calloc(n)
    AV[v]=1
    while(v≠P[v]){
        AV[P[v]]=1
        v=P[v]
    }
    while(w≠P[w]){
        if(AV[w]==1){ return w }
        w=P[w]
    }
}
```

Dato un albero rappresentato come vettore di padri scrivere lo pseudocodice di un algoritmo che trova la distanza dalla radice di ogni nodo dell'albero in $O(n)$

```
Dist_root(P: array){
    Dist[n]= {-1,-1...,-1}
    For(i=0...,n-1){
        if(Dist[i]==-1){ dist(P,i,Dist) }
    }
    return Dist
}

dist(P:array, i:nodo, Dist: array){
    if (P[i]=i){
        Dist[i] = 0
        return 0
    }
    if(Dist[i]!=1){return Dist[i]}
    Dist[i] = dist( P, P[i], Dist)+1
}
```

Dato un grafo $G=(V,E)$, data la funzione $\omega: E \to \{1,2\}$ che associa ad ogni arco $e \in E$ un peso (1 o 2), definiamo la distanza fra due vertici $x,y \in V$ come il cammino con il peso totale minimo.

Scrivere lo pseudocodice di un algoritmo che trova la distanza fra due vertici x,y nel grafo pesato, in $O(n+m)$

Trasformo il grafo, ogni arco di peso 2 diventa un nodo con 2 archi di peso 1


```
Edit_Graph(G:grafo){
    NG = G
    for each( (x,y)∈E(G)){
        if (ω((x,y)) ==2){
            E(NG)=E(NG)\{(x,y)}
            z = new node
            V(NG)=V(NG)U{z}
            E(NG)=E(NG) U {(x,z)}
            E(NG)=E(NG) U {(z,y)}
            ω((x,z))=1
            ω((z,y))=1
        }
    }
    return NG // Basterà Fare una BFS sul nuovo grafo
}
```