Esercizio 1 (22.2-9, [1]). Fornire una algoritmo in pseudo-codice che, dato un grafo non diretto e connesso $G = (V, E)$, trova una passeggiata in G che attraversa tutti gli archi una e una sola volta in ognuna della due direzioni in tempo $\mathcal{O}(|V| + |E|)$.

Faccio una DFS, ogni volta che considero un nodo già visitato, percorro avanti ed indietro l'arco incidente.

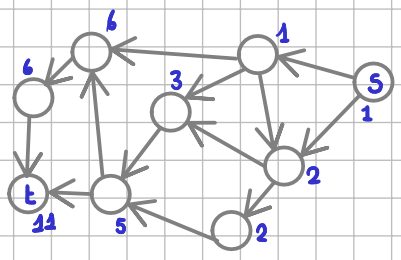
```
DFS(G:grafo, x:nodo, Vis: array, L:lista, prec: nodo){
    if(x≠prec){L.add(prec,x)}
    Vis[x]=1
    for each y~x {
        if(Vis[y]==0){
            DFS(G,y,Vis,L,x)
        }else if(y≠prec){
            L.add(x,y)
            L.add(y,x)
        }
    }
    if(x≠prec){L.add(x,prec)}
}
```

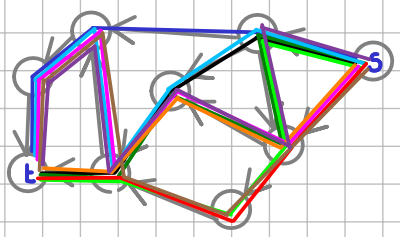
Esercizio 2 (22.4-2, [1]). Fornire un algoritmo in pseudo-codice che dato un grafo diretto e aciclico $G = (V, E)$ e due vertici s e t, restituisce il numero di tutti i cammini da s a t in G.

Sia $\#P(s,t)$ il numero di cammini da s a t, vale la seguente:

$$\#P(s,t) = \sum_{u \in N} \#P(s,u) \quad \text{dove} \quad N = \{u \mid (u,t) \in E(G)\}$$

```
Cammini(s:nodo, t:nodo, G:grafo){
    if(t==s){return 1}
    K=0
    for each u | (u,t)∈E(G){
        K+=cammini(s,u)
    }
    return K
}
```





Esercizio 3. Sia un grafo diretto $G = (V, E)$ ed s e t due vertici di G. G si dice *s-t-connesso* se ogni vertice di G è in almeno un cammino da s a t. Fornire un algoritmo in pseudo-codice che dato un grafo diretto e aciclico $G = (V, E)$ ed s e t due vertici di G, restituisce un sottografo G' di G massimale s-t-connesso.

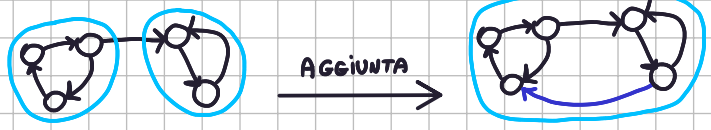
```
ST(G:grafo, s:nodo, t:nodo){
    P:stack
    O:set
    Vis[n] = {0,0...,0}
    Vis[s] = 1
    P.push(s)
    while(s≠0){
        x=P.top()
        if( (x,t)∈E(G) ){
            O=O U P
        }
        if(∃t | Vis[t]=0 ∧ (x,t)∈E(G)){
            Vis[t]=1
            P.push(t)
        }else{P.pop()}
    }
    return O
}
```

Esercizio 4 (22.5-1, [1]). Come cambia il numero di componenti fortemente connesse di un grafo diretto se viene aggiunto un nuovo arco?

- Il numero di componenti può rimanere identico? Si

* Cambia solo se congiunge 2 nodi di componenti differenti

- Il numero di componenti può diminuire? Si



- Il numero di componenti può aumentare? No. Dimostrazione per assurdo:

Sia G un grafo e siano $C = \{C_1, C_2 \dots, C_n\}$ il numero di componenti fort. connesse. Sia (x, t) un nuovo arco, tale che $x \in C_i \wedge t \in C_j \wedge i \neq j$. Sia C_{n+1} una nuova componente da Finire

Esercizio 5 (I. Salvo). Sia $G = (V, E)$ un grafo diretto, un vertice $u \in V$ è detto *principale* se per ogni vertice $v \in V$ esiste un cammino diretto da u a v. Fornire un algoritmo in pseudo-codice che, dato un grafo diretto G, determina tutti i vertici principali di G. E' possibile che tale algoritmo abbia complessità $\mathcal{O}(|V| + |E|)$?

Prop: se u e' principale ed esiste un cammino $v \rightsquigarrow u$, allora v e' principale.

Prop: in un grafo aciclico esiste solo un nodo principale (radice)

L'idea e' quella di ridurre il grafo con i vertici contratti sulle componenti, per poi cercare da lì l'unico nodo principale.

idea 2:

```
DFS_prin(G:grafo, x:nodo, Vis:array, O: array
    Vis[x]=1
    For each (y | ∃(x,y)∈E(G)){
        if(Vis[y]≠1){ DFS_prin(G,y.Vis,O) }
    }
    if(∃i | Vis[i]==0){
        for each (y | ∃(y,x)∈E(G)){
            if(Vis[y]≠1){ DFS_prin(G,y.Vis,O) }
        }
    }
    else{
        O[x]=1
        for each (y | ∃(y,x)∈E(G))
            if(O[y]≠1){ DFS_prin(G,y.Vis,O) }
        }
    }
}
```

i e' principale $\Leftrightarrow$ O[i]=1