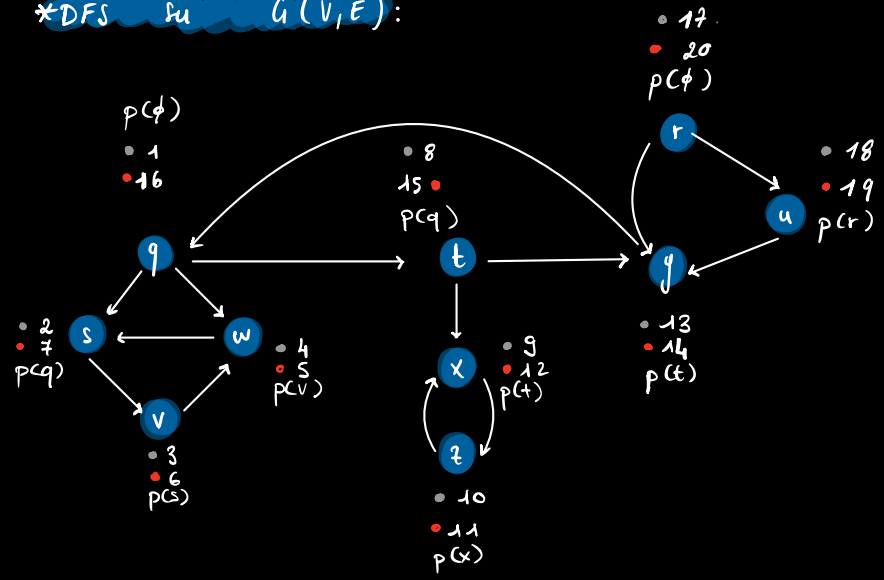
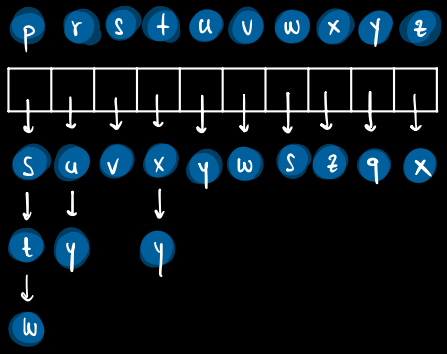
1) ESPLORA IL GRAFO CON DFS: 25/11

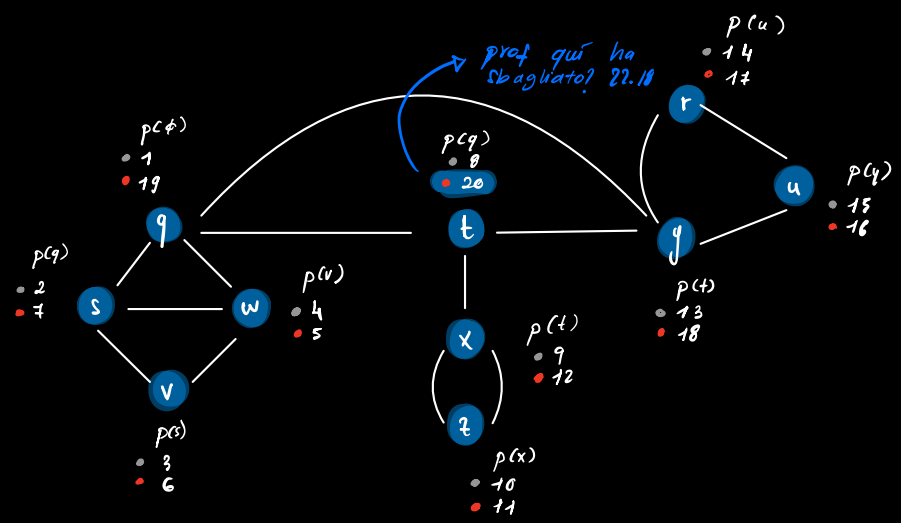
*DFS su G(V,E):



*LISTA DI ADIACENZA:



2) SE FOSSE STATO NON ORIENTATO?



3) MODIFICANDO L'ALGORITMO, VOGLIAMO:

Dato un grafo orientato, vogliamo che venga stampato IL TIPO DI OGNI ARCO.

* Soluzione:

1) Prendo l'algoritmo std dalla lezione precedente.

questa la lascio così.

```
DFS(G)
    For (v ∈ V): {
        color(v) = w
        p(v) = NIL
    }
    Time = 0
    For ( all v ∈ V ){
        if ( color(v) == "white") {
            DFS_VISIT (v)
        }
    }
```

```
DFS_VISIT (G, u)
    time ++
    d[u] = time;
    color[u] = G;
    For all w ∈ Adj[u]:
        if (color[w] = w) {
            p[w] = u;
            DFS_VISIT (G, w);   → Qua parte una visita! Ho un arco
        }
    * → qui potrei aggiungere qualcosa
    color[u] = B;
    time ++;
    F[u] = time;
```

la modifica:

```
DFS_VISIT (G, u):
    time ++
    d[u] = time
    color[u] = G
    For w ∈ Adj[u]: \ p[u]
        IF color(w) == "white":
            p[w] = u;
            print ("Arco Tree!");
            DFS_VISIT (G, w);
        if color(w) == "gray":
            print ("Arco Back!");
        else:
            IF d[u] < d[w]:
                print ("Arco Forward!");
            else
                print ("Arco Cross!");
    color[u] = "Black";
    time ++;
    F[u] = time;
```

piccola ottimizzazione.
Nei grafi non orientati così EVITO DI STAMPARE 2 volte lo stesso arco (prima come Tree, poi come Back).

→ Questo sull'orientato non lo avremo mai.

so che ho un ciclo quando ho un Arco Back. → @ grigio!

4) DFS → grafo orientato, stabilire che G è aciclico

* Modifichiamo l'IF di DFS_visit.

IF color(w) == "white":
    (qui è uguale);

IF color(w) == "grigio":
    Acyclic = False;

* problema: 2 componenti connesse differenti

* se voglio sapere se è TRUE facilmente. Se è aciclico qui non ho problema. Ho FALSE o

* se NON FERMO subito l'algoritmo → faccio to" visite!

lo migliovo per fermarsi prima!

For v∈V
    color[v] = w; p[v] = nil
time = 0; Acyclic = true;
i = 1
while i ≤ n and Acyclic: → bloccante per il booleano Acyclic.
    IF color[v] == w:
        DFS_visit(v).
    i++;

Return Acyclic.

→ Posso intervenire anche sulla DFS_visit(G,u).

* Come? Dato un nodo NON visito tutti i suoi adiacenti. (non sempre)
↳ Anche qui mi fermo prima).

NON È UN CICLO!

DFS_visit(G,u): / P(u) x grafi N.O. → booleano bloccante.
↳ while visit all w ∈ Adj[u] AND Acyclic
↳ blocco. Restituisco al chiamante. Risparmio chiamate DFS_VISIT.

tempi: sempre O(|V|+|E|), ma migliorano i tempi medi.

4) Calcolare il numero di **Componenti Connesse** di $G(V,E)$. MODIFICHIAMO LA DFS.

Semplice → è nella parte DFS(G) che faccio partire le visite in profondità x una comp.

DFS(G)

```
# setting iniziale
For v ∈ V
    color(v) = w
    P[v] = NIL
time = 0; comp_connesse = 0;
```

```
# numero componenti
For v ∈ V
    IF color[v] == w
        DFS_visit(G, v)
        componenti_connesse++;
Return (componenti_connesse);
```

5) Algoritmo che dato un grafo NON O. mi permette di stabilire "a quale componente connessa appartiene ogni vertice".

DFS(G)

```
# setting iniziale
For v ∈ V
    color(v) = w
    P[v] = NIL
time = 0; cc = 1;
```

```
# numero componenti
For v ∈ V
    IF color[v] == w
        DFS_visit(G, v)
        cc++;
```

Ad momento della scoperta di un vertice → segno comp.

↑

```
DFS_visit(G, u)
time++;
d[u] = time;
color[u] = grigio;
# setto la sua componente
com_comp[u] = cc;
.... il resto è uguale ....
```

ACCHIO AL PREDECESSORE!

6) G non orientato e $k > 0$, stabilire se G è una foresta di k Alberi? (k = comp. conn.)

- più componenti (alberi) connessi.
- senza cicli.
- Valuto le cc.

DFS(G, k):

For $v \in V$:

color(v) = W;
P[v] = nil;
time=0; Acyclic = TRUE;
cc = 0;

For all $v \in V \wedge cc \leq k \wedge$ Acyclic → interrompo appena False

If color[v] == W AND cc < k :

DFS_VISIT (G, v);
cc++;

else_if color[v] = W :

cc++;

if cc = k and Acyclic

return TRUE;

else

return FALSE;

∀ Ancora → blocco appena False
*

DFS_VISIT (G, u)

time++;
d[u] = time;
color[u] = grigio;

Serve a non ripercorrere il genitore.

For all $w \in Adj[u] - P(u)$ AND Acyclic.

If color[w] = "W"

P[w] = u
DFS-VISIT (G, w);

else

Acyclic = FALSE.

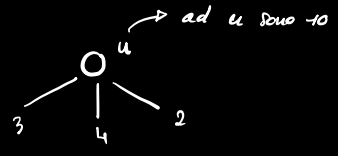
color[u] = "Black";
time++
f[u] = time;

7) Dato $G(V,E)$ N.O. → quante comp. connesse contengono un # pari di vertici?

* **TRUCCO!** → numero di nodi = tempo inizio della componente - tempo di fine. = quantità nodi visitati!

```
DFS(G)
Pari_cc = 0;
For v ∈ V
    Color[v] = W;   p[v] = nil
time = 0;
For v ∈ V
    If color[v] == "white" :
        n_v = DFS_VISIT(u);
        If n_v %. 2 == 0
            cc_pari++;
Return cc_pari;
```

```
DFS_VISIT(G, u):
time ++;
d[u] = time;
color[u] = "Gray";   count = 0;
For w ∈ Adj[u]:
    IF color(w) == "white"
        count = count + DFS_VISIT(G, w);
color[u] = "Black";
time++;
F[u] = time;
count++;
return (count);
```



Prox. lez: 19/11 Al Recupero.