DFS

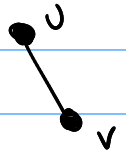
• depth-first search

• visita in profondità

- la visita in profondità si basa sull'idea di
  - appena un vertice viene SCOPERTO allora si completa la visita di quel vertice.

Similarità con la visita in ampiezza

- vertici (u) sconosciuti (u) scoperti (u) completato
- un vertice viene scoperto e analizzato una sola volta quindi passa da BIANCO a GRIGIO e da GRIGIO a NERO solo UNA volta
- ALBERO di VISITA DF: quando un vertice v viene scoperto durante l'analisi del VICINATO di u, allora



è un arco che viene aggiunto all'albero di visita

DIFFERENZA CON BFS

\ Se durante l'ispezione del vicinato di u si scopre un vertice v allora si esegue IMMEDIATAMENTE la visita di v

\ L'albero di visita viene costruito in profondità invece che in ampiezza

color = ["white", ... , "white"]
parent = [None, ... , None]

```
DFS(G, s)
    color[s] ← "gray"
    for u in G[s]:
        if color[u] == "while":
            parent[u] ← s
            DFS(G, u)
    color[s] ← "black"
```

- Lo schema è ricorsivo
- STRUTTURA
  LIFO , last-in first-out
- L'ultimo vertice scoperto è il primo ad essere visitato

BFS = usa una coda

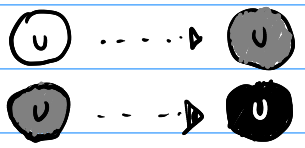
DFS = usa una pila (implicitamente)

VISUALIZZA ESEMPIO PRECALCOLATO

OSS I cammini trovati NON sono i più brevi

- Due aggiunte

1- inseriamo un OROLOGIO. All'inizio è settato al 1 e si incrementa ogni volta che
   - un vertice viene scoperto (e visitato)
   - un vertice viene completato



Manteniamo per ogni vertice il momento in cui questi eventi accadono

2- come per BFS, consideriamo un algoritmo che alla fine di ogni visita passa al primo vertice ancora non scoperto

VISUALIZZA ESEMPIO PRECALCOLATO

- DFS da un vertice

```python
time   = 0
inizio = [None]*len(G)
fine   = [None]*len(G)
P      = [None]*len(G)
color  = ["white"]*len(G)

def DFSvisit(G,s):
    time      = time + 1
    inizio[s] = time
    color[s]  = "gray"

    for u in G[s]:
        if color[u] == "white":
            P[u] = s
            DFSvisit(G,u)

    color[s] = "black"
    time     = time + 1
    fine[s]  = time
```

VARIABILI CHE MANTENGONO LO STATO TRA TUTTE LE CHIAMATE

INIZIO : BIANCO → GRIGIO

FINE : GRIGIO → NERO

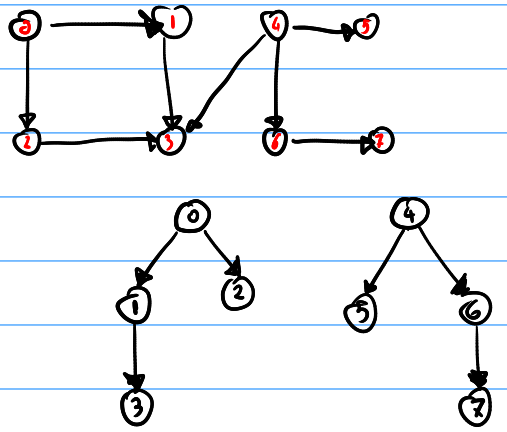
- DFS che esplora tutto il grafo e crea più alberi

```python
def DFS(G):

    time   = 0
    inizio = [None]*len(G)
    fine   = [None]*len(G)
    P      = [None]*len(G)
    color  = ["white"]*len(G)

    for s in range(len(G)):
        if color[s] == "white":
            DFSvisit(G,s)

```



Complessità di DFS è $O(|V|+|E|)$

- ogni vertice è visitato una volta sola
- per ogni vertice visitato si esplora la sua lista di adiacenza

$$\underbrace{\sum_{v \in V} O(1)}_{O(|V|)} + \underbrace{O(\text{outdegree}(v))}_{O(\sum_v \text{outdegree}(v)) = O(|E|)}$$

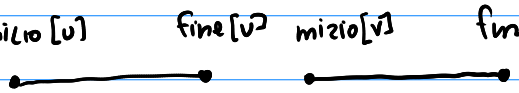
Caratteristica Fondamentale della DFS

- la relazione tra i valori di inizio e fine e il fatto di essere discendenti in uno degli alberi costruiti alla visita DF

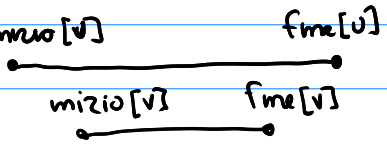
Thm (Teorema 22.7, Teorema delle parentesi)

- Dati due vertici U e V nel grafo (diretto o semplice)

Ci sono 3 casi

(A) 

U e V non sono discendenti uno dell'altro

(B) 

U ↓ V nell'albero DF

(C) viceversa... di (B)

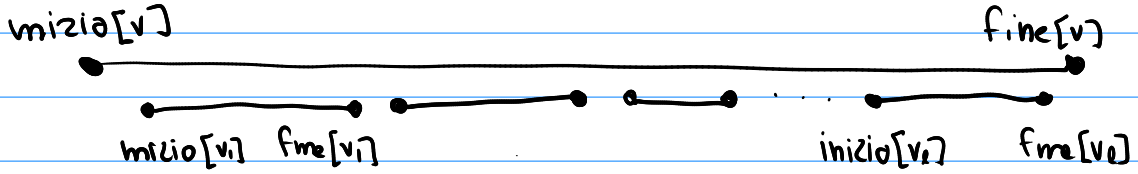
In particolare non può accadere che

oss Ogni vertice è visitato esattamente una volta.

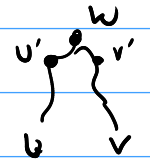
La visita ha luogo nell'intervallo inizio[v] — fine[v]

OSS 1 le visite fatte nei cicli alle righe 12-15 e 29-31 a pag (3) sono su intervalli disgiunti

OSS 2 Se nel ciclo 12-15 nella visita di un vertice V vengono chiamate le visite per $v_1, v_2, \dots v_k$ allora



- Se due vertici sono discendenti uno dell'altro, OSS2 implica intervalli annidati

- Se due vertici non sono discendenti
  - due alberi diversi → intervalli disgiunti
  - stesso albero: 

    u discendente di u' ; v discendente di v' → u' e v' intervalli disgiunti per OSS2
  - u discendente di u' → inizio[u'] < inizio[u] < fine[u] < fine[u']
  - v discendente di v' → lo stesso
  - quindi di intervalli di u e v sono disgiunti

- CLASSIFICAZIONE DEGLI ARCHI

  - Dato $G=(V,E)$ semplice o diretto

applicando la DFS otteniamo una foresta di alberi DF

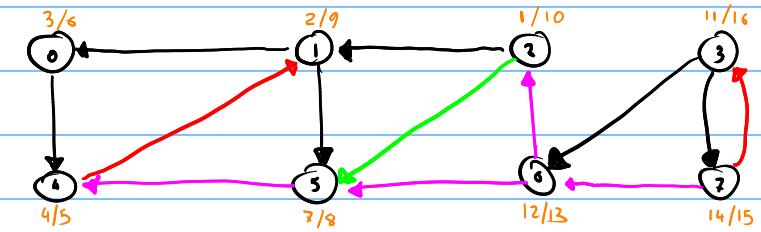
- Gli archi di G possono essere di 4 tipi

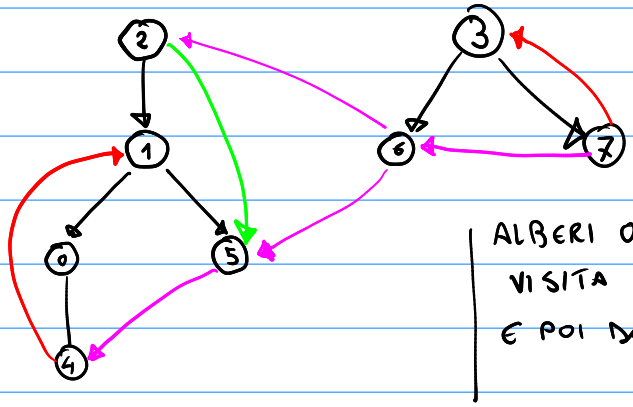
- ARCHI DELL'ALBERO

- ARCHI IN AVANTI
  FORWARD
  da un vertice ad un suo discendente

- ARCHI ALL'INDIETRO
  BACKWARD
  da un vertice da un suo antenato

- ARCHI TRASVERSALI
  CROSSING
  tra vertici che non sono in relazione di antenato-discendente

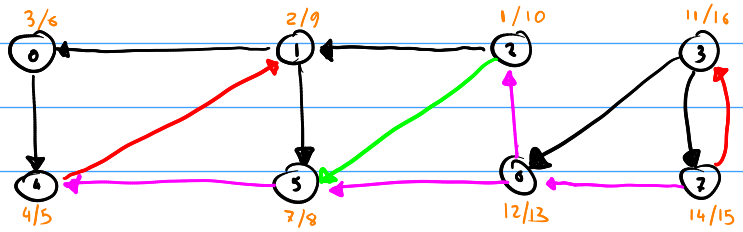




ALBERI OTTENUTI FACENDO VISITA DAL VERTICE 2 E POI DAL VERTICE 3

IN UN GRAFO DIRETTO:

- consideriamo u ed un suo vicino v
- durante la visita di u si cicla sui vicini di u.

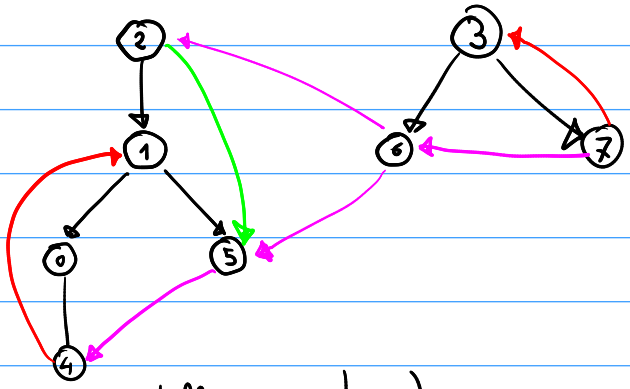


- Se v è BIANCO

  u ⟶ v è un arco dell'albero

- Se v è GRIGIO

  u ⟶ v è un arco all'indietro



- Se v è NERO

abbiamo solo 2 casi possibili (vedi teorema delle parentesi)

① inizio[u] < inizio[v] < fine[v] (a questo punto fine[u] è ancora −)

u ⟶ v un arco in avanti

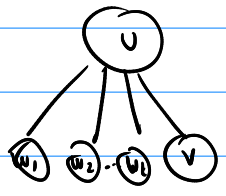
② inizio[v] < fine[v] < inizio[u]

u ⟶ v un arco crossing

IN UN GRAFO SEMPLICE

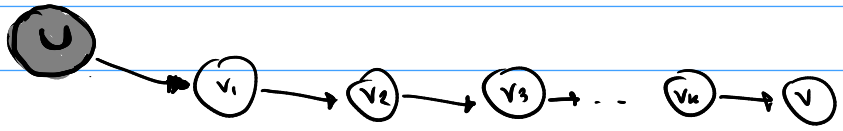
- gli archi possono solo essere
  - archi d'albero
  - archi all'indietro (questo è il teorema 22.10)

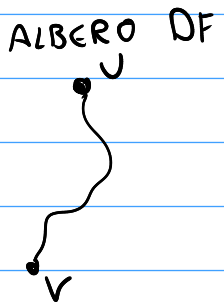
oss Consideriamo {u,v} con inizio[u] < inizio[v] (ovvero u scoperto prima di v)



- se dopo la visita di w_i il vertice v è ancora bianco allora u—v è un arco dell'albero BF (vedi algoritmo)
- se v viene scoperto durante la visita di un w_i, allora v diventa nero. Durante la visita di v, l'arco {u,v} viene ispezionato mentre u è grigio. Quindi l'arco è rosso.

Thm Teorema del cammino bianco



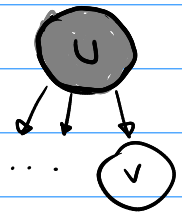


- Un vertice v è discendente di un vertice u nell'albero DF

se e solo se

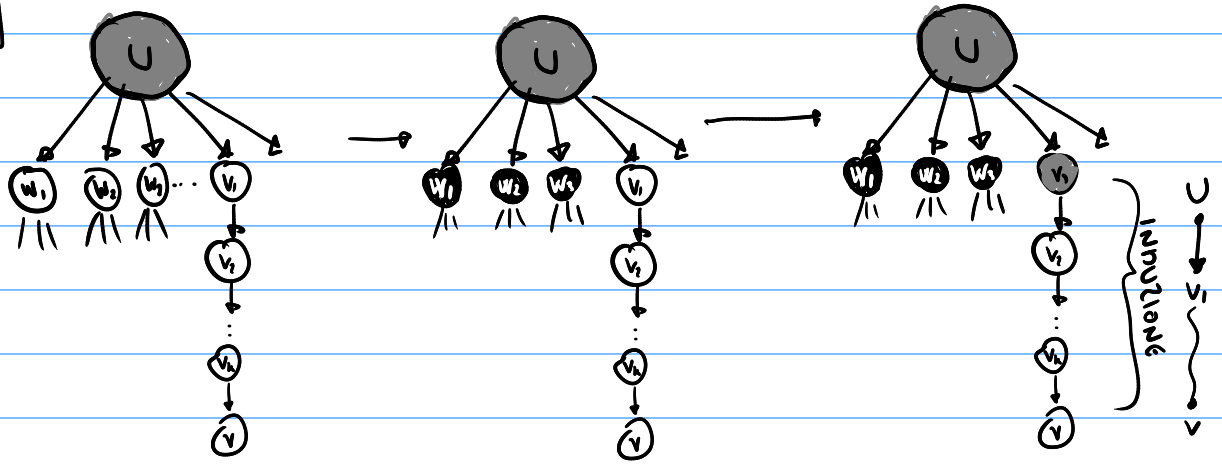
- quando u viene scoperto durante una visita, il vertice v è raggiungibile da u attraverso un cammino di vertici bianchi

dim Caso base

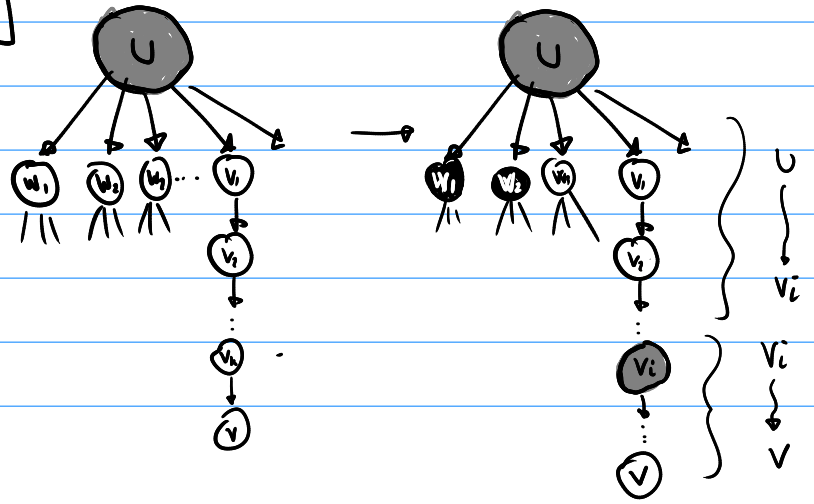


- inizio[u] < inizio[v] e inizio[v] < fine[u]
- quindi gli intervalli sono annidati e quindi u ⇝ v nell'albero DF

Passo Ind 1



Passo Ind 2



ESERCIZIO 22.3-3

Dimostrate che l'arco (u,v) è

- un arco dell'albero o un arco in avanti (forward)
  se e solo se
  $inizio[u] < inizio[v] < fine[v] < fine[u]$

- un arco all'indietro (backward) se e solo se
  $inizio[v] \leq inizio[u] < fine[u] \leq fine[v]$ (anche i loop)

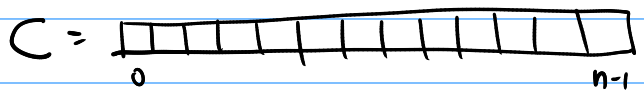
- un arco trasversale (crossing) se $inizio[v] < fine[v] < inizio[u] < fine[u]$

ESERCIZIO 22.3-7

Riscrivete DFS(G,v) eliminando la ricorsione

ESERCIZIO 22.3-12

Modificate DFS(G) in modo tale su G non orientato, calcoli



dove $C[u] = C[v]$ se e solo se u e v sono nella stessa componente connessa.