# Componenti fortemente connesse di un grafo orientato

Corso Ottimizzazione Combinatoria (IN440)

Prof. M. Liverani

# Cammini di costo minimo

- Dato un grafo casuale $G = (V, E)$ orientato con $n$ vertici, si vogliono trovare le componenti fortemente connesse del grafo, ossia i sottografi $G_1, G_2, \ldots, G_k \subseteq G$ tali che per ogni coppia di vertici $u, v \in V(G_h)$ esistono due cammini $p_1 : u \rightsquigarrow v$ e $p_2 : v \rightsquigarrow u$ in $G_h$
- Utilizzeremo l'**algoritmo di Kosaraju-Sharir**, consiste nell'eseguire una visita in profondità del grafo $G$ e una successiva visita in profondità del grafo trasposto $G^T$, in cui gli spigoli sono invertiti, scegliendo i vertici nell'ordine di fine visita ottenuto durante la prima visita in profondità

**Algoritmo 14** COMPONENTIFORTEMENTECONNESSE($G$)

**Input:** Il grafo $G$ orientato

**Output:** Le componenti fortemente connesse di $G$

1: DFS(G) e calcola i tempi finali $f_v$ per ogni $v \in V(G)$
2: calcola $G^T$
3: esegui DFS($G^T$), ma nel ciclo principale considera i vertici in ordine decrescente rispetto a $\{f_v\}$
4: ciascun albero della foresta prodotta al passo precedente è una componente fortemente connessa di $G$

# Visita in profondità del grafo

- Per la visita in profondità del grafo utilizziamo l'algoritmo ricorsivo standard
- Nella implementazione in Python useremo i metodi `u.setColor(...)` e `u.getColor(...)` per marcare i vertici durante la visita
- Useremo invece i metodi `u.setDistance(...)` e `u.getDistance(...)` per gestire i tempi di fine visita

**Algoritmo 55** DFS($G$)

**Input:** Il grafo $G = (V, E)$
**Output:** I tempi di fine visita $t_u$ di ogni vertice $u \in V(G)$

```
1: per ogni u ∈ V(G) ripeti
2:     colore(u) = bianco
3: fine-ciclo
4: t = 0
5: per ogni u ∈ V(G) ripeti
6:     se colore(u) = bianco allora
7:         aggiungi il vertice u alla foresta T: V(T) = V(T) ∪ {u}
8:         VISITA(G, u, t)
9:     fine-condizione
10: fine-ciclo
```

**Algoritmo 56** Visita($G, u, t$)

```
1: colore(u) = grigio
2: per ogni v ∈ N(u) ripeti
3:     se colore(v) = bianco allora
4:         aggiungi il vertice v e lo spigolo (u, v) alla foresta T
5:         VISITA(G, v, t)
6:     fine-condizione
7: fine-ciclo
8: colore(u) = nero
9: t = t + 1, t_u = t
```

# Visita in profondità del grafo

- Per la visita in profondità del grafo utilizziamo l'algoritmo ricorsivo standard
- Nella implementazione in Python useremo i metodi `u.setColor(...)` e `u.getColor(...)` per marcare i vertici durante la visita
- Useremo invece i metodi `u.setDistance(...)` e `u.getDistance(...)` per gestire i tempi di fine visita

```
def DFS(G):
  for u in G:
    u.setColor("bianco")
  t = 0
  x = 0
  for u in G:
    if u.getColor() == "bianco":
      x = x + 1
      visita(G, u, t, x)
      t = u.getDistance() + 1
  return
```

```
def visita(G, u, t, x):
  u.setColor("grigio")
  for v in u.getConnections():
    if v.getColor() == "bianco":
      visita(G, v, t, x)
      t = v.getDistance() + 1
  u.setColor(x)
  u.setDistance(t)
  return
```

$x$ è l'identificativo progressivo del sotto-grafo in cui inseriamo il vertice $u$. Lo memorizziamo come «colore» di $u$

# Creazione del grafo trasposto

- Dato il grafo $G$ dobbiamo creare un altro grafo $G^T$ tale che $V(G) = V(G^T)$ e se $v$ è adiacente a $u$ in $G$, allora aggiungiamo $(v, u)$ come spigolo di $G^T$

**Algoritmo 57** transposeGraph$(G)$

**Input:** Il grafo $G = (V, E)$
**Output:** Il grafo trasposto $G^T = (V, E')$

```
1: per ogni u ∈ V(G) ripeti
2:     per ogni v adiacente a u ripeti
3:         E(G^T) = E(G^T) ∪ {(v, u)}
4:     fine-ciclo
5: fine-ciclo
6: return G^T
```

# Creazione del grafo trasposto

- Dato il grafo $G$ dobbiamo creare un altro grafo $G^T$ tale che $V(G) = V(G^T)$ e se $v$ è adiacente a $u$ in $G$, allora aggiungiamo $(v, u)$ come spigolo di $G^T$

```
def transposeGraph(G1, G2):
  for u in G1:
    G2.addVertex(u.getId())
  for u in G1:
    for v in u.getConnections():
      G2.addEdge(v.getId(), u.getId())
  return
```

# Vertici in ordine decrescente di tempo di fine visita

- Dopo aver visitato il grafo $G$ e calcolato così i tempi di fine visita di ogni vertice, nella visita in profondità del grafo trasposto di $G$ dobbiamo considerare il vertici **in ordine decrescente di tempo di fine visita**
- Se i vertici sono $V = \{1, 2, ..., n\}$ anche i tempi di fine visita sono $t = \{0, 1, ..., n-1\}$
- Per far questo costruiamo una lista con i vertici in ordine decrescente di fine-visita: indichiamo con $t_v$ il tempo di fine visita del vertice $v$, allora

$$f[n - t_v - 1] = v$$

è un array con i vertici di $G$ in ordine decrescente di tempo di fine visita: $f[i]$ è il vertice $v$ con tempo di fine visita $t_v = i$

# Mettiamo insieme i pezzi

```
from in440 import *

def visita(G, u, t, x):
  u.setColor("grigio")
  for v in u.getConnections():
    if v.getColor() == "bianco":
      visita(G, v, t, x)
      t = v.getDistance() + 1
  u.setColor(x)
  u.setDistance(t)
  return

def DFS(G):
  for u in G:
    u.setColor("bianco")
  t = 0; x = 0
  for u in G:
    if u.getColor() == "bianco":
      x = x + 1
      visita(G, u, t, x)
  t = u.getDistance() + 1
  return

g  = Graph()
g1 = Graph()
n  = int(input("Numero di vertici: "))
p  = float(input("Probabilita': "))
```

```
[...segue...]
randomDiGraph(g, n, p)
printGraph(g)
DFS(g)
print("Vertici e tempi di fine visita:")
for u in g:
  print(u.getId(), ":", u.getDistance())
transposeGraph(g, g1)
f = np.empty((len(g1.getVertices())), dtype=int)
for u in g:
  f[n-u.getDistance()-1] = u.getId()
for u in g1:
  u.setColor("bianco")
t = 0
x = 0
for i in range(n):
  u = g1.getVertex(f[i])
  if u.getColor() == "bianco":
    x = x + 1
    visita(g1, u, t, x)
    t = u.getDistance() + 1
print("Vertici e componente:")
for u in g1:
  print(u.getId(), ":", u.getColor())
plotGraph(g, "G", "orientato", False)
plotGraph(g1, "G trasposto", "orientato", True)
```