Applicazioni della visita in profondità – Parte III

Algoritmi 2 – Lezione 6

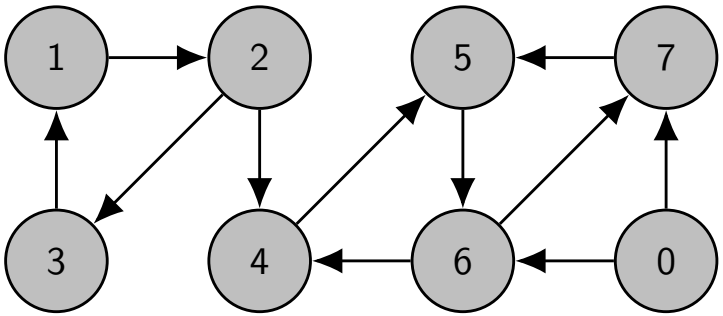
A. Monti
Sapienza Università di Roma

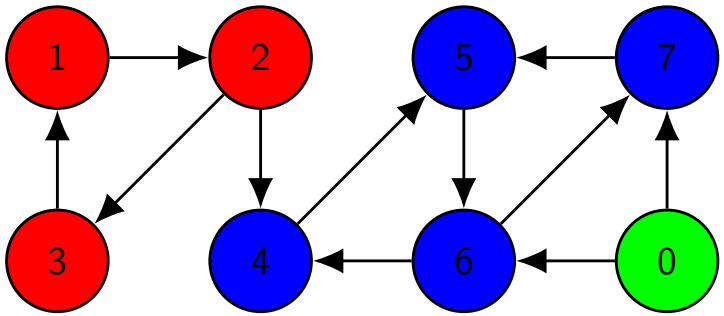
Corso: Algoritmi 2

Indice

Componenti fortemente connesse (SCC)

Una **componente fortemente connessa (SCC)** di un grafo orientato è un sottoinsieme massimo di vertici tale che per ogni coppia di vertici u e v appartenenti alla componente, esiste un cammino orientato da u a v ed un cammino orientato da v a u.

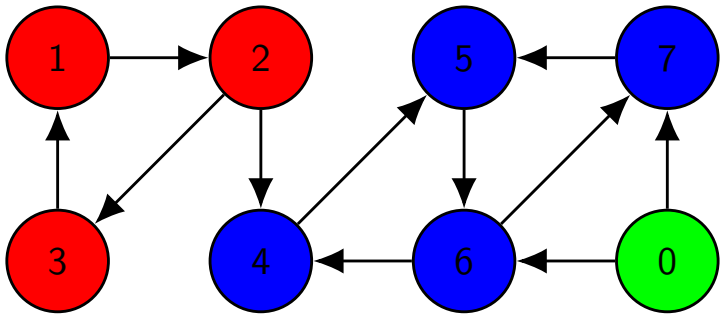




Il grafo G ha 3 componenti fortemente connesse: $\{1, 2, 3\}$, $\{4, 5, 6, 7\}$ e $\{0\}$ evidenziate con colori distinti.

Vettore delle Componenti

- Le componenti fortemente connesse partizionano i nodi del grafo.
- Per identificare rapidamente la componente di ciascun nodo, si usa un **vettore delle componenti fortemente connesse**.
- Questo vettore, chiamato SCC, ha dimensione n (numero di nodi).
- Per ogni nodo i, $SCC[i]$ memorizza un identificatore unico della sua componente.



Un possibile vettore per le tre componenti fortemente connesse del grafo è

$$SCC = \boxed{3}\boxed{1}\boxed{1}\boxed{1}\boxed{2}\boxed{2}\boxed{2}\boxed{2}$$

Trovare una singola componente in $O(n+m)$

- Dato un grafo diretto G e un nodo u, questo algoritmo trova tutti i nodi appartenenti alla componente fortemente connessa di u.
- L'algoritmo si basa su 4 passi:
  1. Calcola l'insieme A dei nodi raggiungibili da u in G.
  2. Calcola l'insieme B dei nodi che in G portano a u.
  3. Restituisce l'intersezione di A e B.
- Per eseguire efficientemente il passo 2, si usa il **grafo trasposto** G^T, ottenuto invertendo la direzione di tutti gli archi di G. I nodi che portano a u in G sono gli stessi che possono essere raggiunti da u in G^T.
- La complessità dell'algoritmo è $O(n+m)$.

Codice: algoritmo per singola SCC

```
def ComponenteFC(G, x):
    '''
    Restituisce la lista dei nodi presenti nella
    componente fortemente  connessa di G che contiene
    il nodo x.
    '''
    n = len(G)
    # Visita i nodi raggiungibili da x nel grafo originale
    visitati1 = [0] * n
    DFS(G, x, visitati1) # Richiede O(n+m)
    # Costruisci il grafo trasposto
    GT = grafoTrasposto(G) # Richiede O(n+m)
    # Visita i nodi raggiungibili da x nel grafo trasposto
    visitati2 = [0] * n
    DFS(GT, x, visitati2) # Richiede O(n+m)
    # Intersezione dei due insiemi di nodi visitati
    Componente = [ ]
    for i in range(n):
        if visitati1[i] == 1 and visitati2[i] == 1:
            Componente.append(i)
    return Componente
```

Analisi della complessità

- **Complessità dell'algoritmo:** Tutte le operazioni (visite DFS, costruzione del grafo trasposto e calcolo dell'intersezione) hanno un costo lineare in $O(n + m)$.

- **Attenzione:** Un algoritmo che costruisce il vettore SCC applicando questa procedura fino a che tutti i nodi risultano assegnati ad una componente avrebbe una complessità di $O(n(n + m))$, ovvero $\Theta(n^3)$ nel caso peggiore, risultando troppo inefficiente.

- Un esempio in cui l'algoritmo ha complessità $\Theta(n^3)$ è il DAG completo: per ogni coppia di nodi (u, v) con $u < v$ esiste un arco $u \to v$.

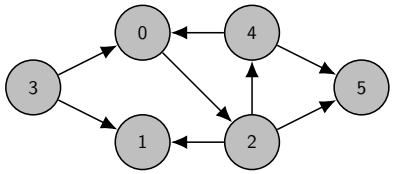
Kosaraju: calcolo di tutte le SCC in $O(n+m)$"

- L' **Algoritmo di Kosaraju** è un metodo efficiente per trovare tutte le componenti fortemente connesse di un grafo orientato.
- Ha una complessità ottimale $O(n+m)$.
- Funziona in due fasi principali, entrambe basate sulla visita in profondità (DFS), la prima visita sul grafo G e la seconda sul grafo G^T.

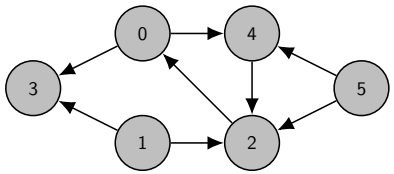
Fasi dell'algoritmo

1. Si esegue una **prima DFS** sull'intero grafo G, annotando il tempo di fine visita per ciascun nodo.
2. Si costruisce il **grafo trasposto** G^T.
3. Si esegue una **seconda DFS** ma su G^T considerando i nodi in ordine decrescente rispetto al tempo di fine visita. Ogni nuova chiamata DFS su un nodo non ancora visitato identifica una nuova SCC del grafo originale G.

Esecuzione passo passo



I nodi di G ordinati per tempo di fine visita sono $[1, 5, 4, 2, 0, 3]$. Nella seconda fase si visita il grafo trasposto G^T:



La visita su G^T va effettuata esplorando i nodi nell'ordine di fine visita decrescente $[3, 0, 2, 4, 5, 1]$. Risultato delle DFS:

- DFS da 3 produce $scc = \{3\}$
- DFS da 0 produce $scc = \{0, 2, 4\}$
- DFS da 2 già visitato
- DFS da 4 già visitato
- DFS da 5 $scc = \{5\}$
- DFS da 1 $scc = \{1\}$

Le componenti fortemente connesse di G sono $\{3\}$, $\{0, 2, 4\}$, $\{5\}$ e $\{1\}$.

Codice Python: algoritmo di Kosaraju

```
def kosaraju(G):
    ''' Calcola le componenti fortemente connesse di G '''
    n = len(G)
    Visitati = [False] * n
    Ordine = []
    # Prima DFS per ottenere l'ordine di fine visita
    for u in range(n):
        if not Visitati[u]:
            dfs1(u, G, Visitati, Ordine) #Richiede O(n+m)
    # Trasposizione del grafo
    GT = grafoTrasposto(G) # Richiede O(n+m)
    Visitati = [False] * n
    SCC = [0] * n
    c = 0
    Ordine.reverse()
    # Seconda DFS per assegnare componenti
    for u in Ordine:
        if not Visitati[u]:
            c += 1
            dfs2(u, GT, Visitati, c, SCC) # Richiede O(n+m)
    return SCC

def dfs1(u, G, Visitati, Ordine):
    Visitati[u] = True
    for v in G[u]:
        if not Visitati[v]:
            dfs1(v, G, Visitati, Ordine)
    Ordine.append(u)

def dfs2(u, GT, Visitati, c, SCC):
    SCC[u] = c
    Visitati[u] = True
    for v in GT[u]:
        if not Visitati[v]:
            dfs2(v, GT, Visitati, c, SCC)
```

ESERCIZI

Suggerimento: Prima di cercare le soluzioni in rete, prova a ragionarci da solo: gli esercizi che seguono sono pensati per aiutarti a mettere alla prova la tua comprensione, la tua capacità di sviluppare una prova di correttezza e costruire un algoritmo corretto.
Solo dopo averci riflettuto, confronta la tua soluzione con altre possibili versioni.

Esercizi

1. Sviluppare un algoritmo che, dato il grafo orientato G e due suoi nodi u e v, in tempo $O(n+m)$, determina se i due nodi sono mutuamente raggiungibili.

2. Sviluppare un algoritmo che, dato il grafo orientato G, in tempo $O(n+m)$, verifichi se G è un insieme di cicli disgiunti (ad esempio, ogni componente fortemente connessa è un ciclo).

3. Sia G un grafo orientato. Definiamo **cicli indipendenti** di G quei cicli che non sono mutuamente raggiungibili tra loro; in altre parole, non esiste un cammino che permetta di spostarsi da un nodo di un ciclo a un nodo dell'altro e viceversa. Sviluppare un algoritmo che, dato un grafo orientato G, in tempo $O(n+m)$, determini il numero di cicli indipendenti.

Esercizi

4 Sia G un grafo orientato. Definiamo il **grafo condensato** GC come il grafo ottenuto da G sostituendo ogni SCC con un singolo nodo. Esiste un arco in GC da una componente SCC_i a una componente SCC_j se e solo se esiste almeno un arco $(u, v) \in E$ con $u \in \text{SCC}_i$ e $v \in \text{SCC}_j$.

1. Dimostrare che il grafo condensato GC è sempre un grafo diretto aciclico (DAG).
2. Sviluppare un algoritmo che, dato il grafo orientato G, in tempo $O(n + m)$, restituisce il grafo condensato di G (DAG delle componenti fortemente connesse).
3. Sviluppare un algoritmo che, dato il grafo orientato G, in tempo $O(n + m)$, conti il numero minimo di nodi da cui partire per visitare tutti i nodi di G.
4. Sviluppare un algoritmo che, dato il grafo orientato G, in tempo $O(n + m)$, determini il numero minimo di archi da aggiungere per rendere il grafo fortemente connesso.