Esercitazione 2: More Depth-First Search

Giacomo Paesani

March 18, 2025

Esercizio 1 (22.3-11,[1])**.** E' possibile avere un vertice u di un grafo diretto G tale che la foresta ottenuta da una visita in profondità contiene solo u, anche se u ha grado entrante e grado uscente almeno uno?

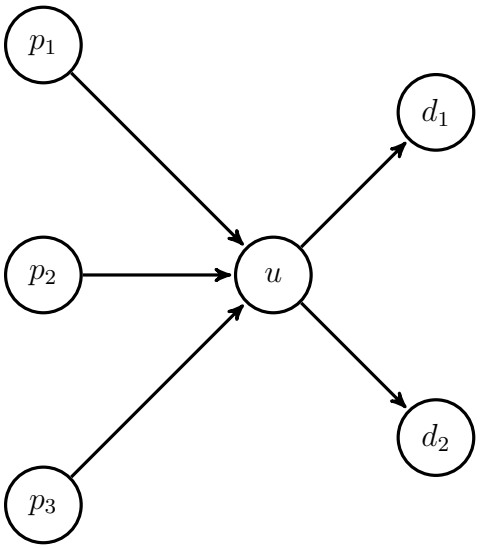
Soluzione 1. Dato un grafo G, la foresta ottenuta da una visita in profondità, e quindi anche la classificazione degli archi in base alla visita, dipende fortemente da come vengono selezionati i vertici del grafo nei vari passaggi dell'algoritmo e da come sono ordinate le liste di adiacenza.

La risposta al esercizio è positiva se si ammette la presenza di cappi. Infatti ogni grafo che ammette una componente connessa contenente il solo vertice u e l'arco (u, u) verifica banalmente le richieste. Dimostriamo ora la seguente affermazione: sia $G = (V, E)$ un grafo diretto e $u \in V$, allora esiste una foresta ottenuta da una visita in profondità tale che u è l'unico vertice della sua componente se e solo se u non appartiene ad alcun ciclo di lunghezza almeno due di G.

$\Leftarrow$ Sia $u \in V$ tale che u non appartiene ad alcun ciclo di G di lunghezza almeno due. Siano $p_1, \dots, p_k$ i vertici di G tali che (p_i, u) è un arco di G, per ogni $i \in \{1, \dots, k\}$, e $d_1, \dots, d_h$ i vertici di G tali che (u, d_j) è un arco di G, per ogni $j \in \{1, \dots, h\}$. Se per qualche $i \in \{1, \dots, k\}$, abbiamo che il vertice p_i viene visitato prima di u, allora p_i e u appartengono allo stesso albero della visita. Se per qualche $j \in \{1, \dots, h\}$, abbiamo che il vertice u viene visitato prima di d_j allora u e d_j appartengono allo stesso albero della visita.

Allora impostiamo le liste di adiacenza in modo che, per ogni $i \in \{1, \dots, k\}$ e per ogni $j \in \{1, \dots, h\}$, u viene visitato prima di p_i e dopo d_j: questo è possibile dato che u non appartiene ad alcun ciclo di lunghezza almeno due. Infatti una volta terminate le visite a partire dai vertici di $\{d_1, \dots, d_h\}$ i vertici $p_1, \dots, p_k$ e u non sono ancora stati visitati. In questo caso u è l'unico

vertice dell'albero di visita che lo contiene, infatti tutti gli archi della forma (p_i, u) e (u, d_j) sono di attraversamento.



$\Rightarrow$ Supponiamo, per assurdo, che u appartiene ad un ciclo $C = \{v_1, \ldots, v_k\}$ con archi (v_i, v_{i+1}) e (v_k, v_1), per $k \geq 2$, con $v_1 = u$. Sia T l'albero di visita di G che comprende un vertice v di C, allora è facile osservare che T visita tutti i vertici della componente fortemente connessa di G che contiene v, inclusi tutti i vertici di C e in particolare u. Dato che $|C| \geq 2$, abbiamo che u non è l'unico vertice visitato in T e quindi una contraddizione. $\square$

Esercizio 2. In un grafo non diretto e connesso $G = (V, E)$ un vertice v si dice di *articolazione* (*cutvertex* in inglese) se $G - v$, il grafo ottenuto da G rimuovendo v e tutti gli archi ad esso incidenti, non è connesso. Modificare l'algoritmo della ricerca in profondità in maniera da poter ottenere tutti e soli i vertici di articolazione di un grafo connesso; è possibile fare questa modifica in modo che il controllo avvenga in $\Theta(|V| + |E|)$?

Come si può ulteriormente modificare l'algoritmo per ottenere tutti i vertici di articolazione di un grafo non necessariamente connesso (dove in questo caso il criterio di un vertice di articolazione è quello di disconnettere la componente connessa che lo contiene)?

Soluzione 2. E' necessario ricordare che nella classificazione degli archi di un grafo non diretto in seguito ad una ricerca in profondità possono essere

presenti solo archi dell'albero o all'indietro. Come prima cosa osserviamo che essendo G non diretto e connesso, ogni ricerca in profondità restituisce un singolo albero.

Prima di dare la soluzione dobbiamo fare alcune osservazioni su come individuare i vertici di articolazione durante la ricerca in profondità. Iniziamo considerando la radice u dell'albero della ricerca, il vertice di G da cui inizia la visita: u è un vertice di articolazione di G se e solo se ci sono due archi dell'albero incidenti ad u. Sia ora u un vertice che non è la radice. Allora u è un vertice di articolazione di G se e solo se non esiste un arco all'indietro da un discendente di u ad un antenato di u.

L'Algoritmo 1 è una delle possibile soluzioni: il codice è inspirato all'algoritmo ricorsivo di ricerca in profondità proposto in [1]. Notiamo come tale algoritmo è stato modificato per fornire l'insieme dei punti di articolazione del grafo. In questa versione dell'algoritmo la funzione **DFS-Visit** con input (G, u) ritorna un intero che indica il minimo tempo di inizio visita tra tutti vertici visitati dai discendenti di u.

La struttura dati coda A, che viene inizializzata come vuota nella linea 2, ha il ruolo di collezionare nel corso dell'algoritmo tutti i vertici di articolazione del grafo in esame. Consideriamo inizialmente la radice u dell'albero; se, una volta terminata la visita, ci sono almeno due archi dell'albero incidenti ad u (e quindi la condizione in linea 34 è soddisfatta) allora u è di articolazione e viene aggiunto alla coda A con la funzione $enqueue(u)$ (linea 35). La variabile *children* è definita per ogni vertice u del grafo (ogni volta che viene chiamata la funzione **DFS-Visit**(G, u)) e viene incrementata (linea 27) ogni volta che da u si visita per la prima volta un'altro vertice v; notiamo però che tale variabile è utilizzata solo per la radice dell'albero.

Sia ora u un vertice diverso dalla radice dell'albero. Per determinare se u è di articolazione o no, è necessario studiare gli archi all'indietro che originano dai discendenti di u. La variabile b (calcolata in linea 28) indica il minimo tempo di visita tra i vertici visitati da v e viene confrontata con il tempo di inzio visita di u: se la condizione in linea 29 è soddisfatta, e cioè se ogni arco all'indietro con origine da un discendante di u ha come termine in u o in un suo discendente, allora u è di articolazione e viene aggiunto all'insieme A in linea 30. Concludiamo notando che la variabile *back* ha la funzione di registrare il minimo tempo di visita tra i vertici visitati da u: è facile verificare che *back* assume valori minori o uguali di $t[u]$ (in linea 21). Questa variabile viene confrontata, e in caso aggiornata, con b e $t[v]$ (linea 31 e linea 33, rispettivamente) ogni volta che viene visitato un vertice a partire

Algorithm 1 DFS modificata per ricavare tutti i punti di articolazione di un grafo non diretto.

Input: grafo non diretto $G = (V, E)$.
Output: l'insieme A dei vertici di articolazione.

```
1: global variables
2:     A ↤ coda, inizialmente vuota
3:     Color ↤ array di |V| elementi
4:     Parent ↤ array dei padri
5:     t ↤ array dei tempi di inizio visita
6:     T ↤ array dei tempi di fine visita
7:     time ↤ intero che simulta il tempo
8: end global variables
9: function DFS(G)
10:    for u ∈ V do
11:        Color[u]=BIANCO
12:    time = 0
13:    for u ∈ V do
14:        if Color[u]==BIANCO then
15:            b=DFS-Visit(G,u)
16:            Parent[u] = u
17:    return A
18: function DFS-Visit(G,u)
19:    time = time + 1
20:    t[u] = time
21:    back= time
22:    Color[u]=GRAY
23:    children=0
24:    for v ∈ Adj[u] do
25:        if Color[v]==BIANCO then
26:            Parent[v] = u
27:            children=children+1
28:            b=DFS-Visit(G,v)
29:            if t[u] > 1 and b ≥ t[u] then
30:                A.enqueue(u)
31:            back=min{back,b}
32:        if Color[v]!=BIANCO and v! = Parent[u] then
33:            back=min{back,t[v]}
34:    if t[u] == 1 and children≥ 2 then
35:        A.enqueue(u)
36:    Color[u]=NERO
37:    time = time + 1
38:    T[u] = time
39:    return back
```

da u, sia se questo vertice viene visitato per la prima volta o è già in visita. La seconda condizione in linea 32 è necessaria per verificare che v non sia il padre di u.

Si può facilmente notare che l'algoritmo appena descritto è in grado di trovare tutti i vertici di articolazione anche per grafi non connessi, cioè quei vertici la cui rimozione disconnette la componente connessa che li contiene o equivalentemente aumenta il numero di componenti connesse del grafo. Infatti, grazie al **for** nella linea 13, la visita in profondità continua finchè termina la visita di tutti i vertici del grafo. □

Esercizio 3. In un grafo non diretto G un cammino *Hamiltoniano* è un cammino P di G che passa per ogni vertice di G esattamente una volta. Provare o confutare la seguente affermazione: G contiene un cammino Hamiltoniano se e solo se può essere prodotto un albero di ricerca di G che è un cammino. Domanda bonus: che succede se invece del cammino Hamiltoniano, si ha un ciclo Hamiltoniano in G?

Soluzione 3. Iniziamo supponendo che esista un cammino Hamiltoniamo P nel grafo G con n vertici. Possiamo *chiamare* i vertici di G seguendo l'ordine dato dal cammino P: cioè $V(G) = \{v_1, \ldots, v_n\}$, dove v_i precede v_{i+1} in P, per ogni $i = 1, \ldots, n-1$. Allora si considerano le liste di adiacenza dei vertici in maniera che per il vertice v_i, il primo adiacente che si trova è sempre v_{i+1} per ogni $i = 1, \ldots, n-1$. Consideriamo ora la ricerca in profondità partendo dal vertice v_1. Per costruzione, l'albero di ricerca di G usa esattamente gli archi di P e quindi tale albero di ricerca è proprio il cammino P.

Supponiamo ora che può essere prodotto un albero di ricerca per una ricerca in profondità di G che è un cammino P. In particolare il cammino P contiene ogni vertice di G esattamente una volta. Allora, P è un cammino Hamiltoniano di G.

Consideriamo invece il caso in cui G ha un ciclo Hamiltoniano C. Si osserva facilmente che C contiene un cammino Hamiltoniano P ottenuto rimuovendo un qualsiasi arco da C. Allora, per quello che abbiamo mostrato prima si può produrre un albero di ricerca di G che è il cammino P.

Il viceversa non è vero. Consideriamo il grafo P non diretto che è composto da un singolo cammino. Una ricerca in profondità di P con vertice iniziale scelto tra i due estremi di P produce nesessariamente il cammino P. E' allo stesso modo chiaro che G non ammette alcun ciclo Hamiltoniano.

Esercizio 4 (22.3-1,[1])**.** Durante una visita in profondità, ad ogni vertice u di un grafo viene associato un colore che varia nel tempo t dell'algoritmo. Se

$t < t[u]$ e cioè il vertice u deve ancora essere visitato per la prima volta allora u è colorato di BIANCO. Se $t[u] \leq t \leq T[u]$ e cioè il vertice u è ancora in visita allora u è colorato di GRIGIO. Infine, se $t > T[u]$ e cioè il vertice u è stato già completamente visitato allora u è colorato di NERO. Fare una tabella 3x3 con righe e colonne contrassegnate da BIANCO, GRIGIO e NERO. In ogni cella (i, j), indicare se, in un alcun punto di una ricerca in profondità di un grafo diretto, è possibile avere un arco da un vertice di colore i ad un vertice di colore j. Per ogni possibile arco, indicare se esso può essere:

- arco dell'albero,
- arco all'indietro,
- arco in avanti o
- arco di attraversamento.

Fare una seconda tabella per una ricerca in profondità di un grafo non diretto.

	BIANCO	GRIGIO	NERO
BIANCO			
GRIGIO			
NERO			

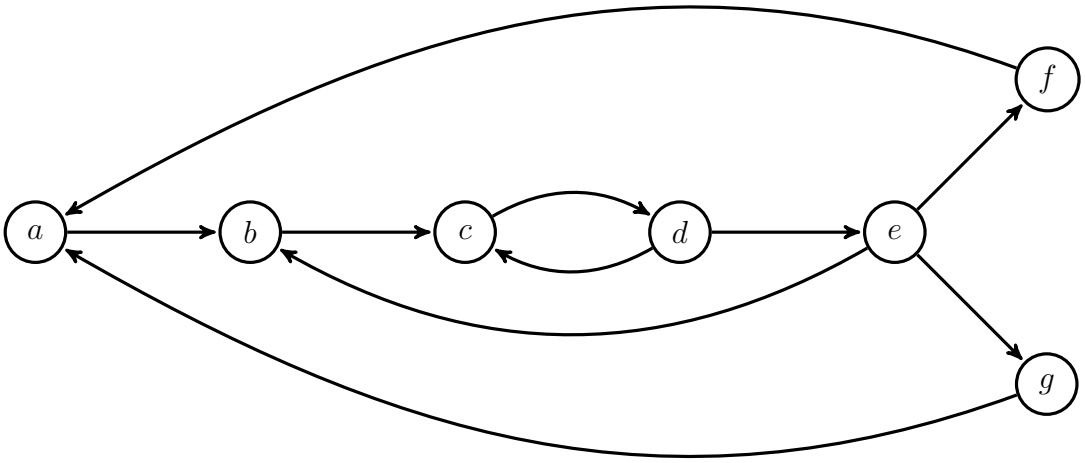
Soluzione 4. Per il caso di un grafo diretto avremo:

	BIANCO	GRIGIO	NERO
BIANCO	Alb Die Ava Att	Die Att	Att
GRIGIO	Alb Ava	Alb Die Ava	Alb Ava Att
NERO	n/a	Die	Alb Die Ava Att

Invece, per il caso di un grafo non diretto avremo:

	BIANCO	GRIGIO	NERO
BIANCO	Alb Die	Alb Die	n/a
GRIGIO	Alb Die	Alb Die	Alb Die
NERO	n/a	Alb Die	Alb Die

Esercizio 5 (I. Salvo)**.** Sia G il grafo raffigurato in figura. Determinare il minimo numero di archi che devono essere eliminati da G affinchè G ammetta ordinamenti topologici. Una volta rimosso questo insime minimo di archi, determinare tutti gli ordinamenti topologici di G.



Soluzione 5. Ricordiamo che un grafo diretto ammette un ordinamento topologico se e solo se è aciclico. Sia C il ciclo di lunghezza due formato dagli archi (c, d) e (d, c). Questo ci permette di trarre due conclusioni. La prima è che è necessario rimuovere almeno un arco da G per renderlo aciclico. Inoltre, ogni sottoinsieme di archi A la cui rimozione rende G senza cicli deve contenere almeno uno tra (c, d) e (d, c). Supponiamo che A contiene l'arco (d, c); in questo caso è facile notare che esiste almeno un ciclo C' di G che non contiene l'arco (d, c), ad esempio quello formato dagli archi (b, c), (c, d), (d, e) e (e, b). Allora A deve anche contenere almeno uno degli archi di C' e quindi $|A| \geq 2$. E' altrettanto immediato da osservare che se $A = \{(c, d)\}$, allora $G - A$, il grafo ottenuto da G rimuovendo gli archi di A, è aciclico. Tutti i possibili ordinamenti topologici sono:

- $[d, e, f, g, a, b, c]$
- $[d, e, g, f, a, b, c]$ □

References

[1] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. Introduction to algorithms. 2022.