Esercitazione 1: Depth-First Search

Giacomo Paesani

March 3, 2024

Esercizio 1. Un grafo diretto $G = (V, E)$ si dice diretto aciclico (DAG) se G non contiene alcun ciclo diretto. Modificare l'algoritmo della ricerca in profondità in maniera da poter controllare se un grafo diretto è diretto aciclico o no, e fornire o un ordinamento topologico o un ciclo del grafo; è possibile fare questa modifica in modo che il controllo avvenga in $\Theta(|V| + |E|)$?

Soluzione 1. Per risolvere questo esercizio è necessaria la seguente affermazione: un grafo diretto è aciclico se e solo se in un qualsiasi albero di ricerca in profondità c'è un arco in indietro. Inoltre, se G è un grafo diretto aciclico, ogni ricerca in profondità fornisce un ordinamento dei vertici di G in base ai tempi di fine visita.

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 non solo per controllare se il grafo diretto in esame è aciclico o no, ma anche per fornire una prova di questo fatto. Supponiamo che l'algoritmo trova un arco (u, v) in indietro (linea 22), cioè in cui $v.d < u.d < u.f < v.f$, allora nelle linee 23 e 24 viene generato il ciclo C sotto forma di una coda (*queue* in inglese) che esprime l'ordine in cui appaiono i vertici. A questo proposito la routine ricorsiva DFS-Path serve ad ottenere l'unico cammino da v ad u che usa gli archi dell'albero di ricerca sotto forma di coda. Il ciclo C, che ottiene dal cammino ricavato con DFS-Path(G,v,u) aggiungendo u in fondo, se viene creato, viene anche riportato dall'algoritmo principale (linea 11).

Finalmente consideriamo il caso in cui la condizione presente nella linea 22 non si verifica mai. Allora ogni vertice u viene inserito nella lista T (linea 29) in base al tempo di chiusura $u.d$ e ciò individua una ordinamento topologico: l'operazione di lista *append(u)* permette di inserire u come l'elemento in testa alla lista.

Algorithm 1 DFS modificata per controllare se un grafo diretto è aciclico o no.

Input: grafo diretto G.
Output: un ordine topologico se G è aciclico e un ciclo altrimenti.

```
1: function DFS(G)
2:     for u ∈ G.V do
3:         u.c=BIANCO
4:     t = 0
5:     C ↤ queue, inizialmente vuota
6:     T ↤ list, inizialmente vuota
7:     for u ∈ G.V do
8:         if u.c==BIANCO then
9:             DFS-Visit(G,u)
10:            if C not empty then
11:                return C
12:    return T
13: function DFS-Visit(G)
14:    t = t + 1
15:    u.d = t
16:    u.c=GRAY
17:    for v ∈ G.Adj[u] do
18:        if v.c==BIANCO then
19:            v.π = u
20:            DFS-Visit(G,v)
21:        else
22:            if v! = u.π and v.c==GRIGIO then
23:                DFS-Path(G,v,u)
24:                C.enqueue(u)
25:                return
26:    u.c=NERO
27:    t = t + 1
28:    u.f = t
29:    T.append(u)
30:    return
31: function DFS-Path(G,v,u)
32:    C.enqueue(u)
33:    if u! = v then
34:        DFS-Path(G,v,u.π)
35:    return
```

Esercizio 2. Un grafo $G = (G, E)$ non diretto dice bipartito se l'insieme dei vertici V può essere partizionato in due insiemi disgiunti U e W tali che: (1) $U \cap W = \emptyset$, (2) $U \cup W = V$ e (3) ogni arco di G è incidente ad un vertice di U e ad vertice di W. E' noto che un grafo G è bipartito se e solo se G non ha cicli di lunghezza dispari. Modificare l'algoritmo della ricerca in profondità in maniera da poter controllare se un grafo non diretto è bipartito o no, e in caso fornire una bipartizione; è possibile fare questa modifica in modo che il controllo avvenga in $\Theta(|V| + |E|)$? Domanda bonus: nel caso G non è bipartito, come deve essere ulteriormente modificato l'algoritmo per ritornare un ciclo dispari di G?

Soluzione 2. L'idea principale è di assegnare un valore, 0 o 1, ad ogni vertice nel momento che viene visitato per la prima volta in maniera che tale valore sia diverso dal valore del padre nella ricerca. Se durante lo svolgimento dell'algoritmo si crea un arco che ha i due estremi con lo stesso valore, allora questo fatto certifica l'esistenza di un ciclo di lunghezza dispari nel grafo e quindi che non è bipartito.

L'Algoritmo 2 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 non solo per controllare se il grafo non diretto in esame è bipartito o no, ma anche per fornire una prova di questo fatto. Dato un vertice u, l'attributo $u.\sigma$ è un valore nell'insieme $\{0, 1\}$ che indica la parità o disparità della lunghezza del cammino dal vertice di partenza della ricerca ad u nel albero di ricerca. Per il vertice di partenza della ricerca z, $z.\sigma$ è inizializzato come 0 in maniera arbitraria (linea 11). Sia ora v un vertice diverso da quello di partenza. Se v viene visitato per la prima volta a partire da un nodo u allora si pone $v.\sigma = 1 - u.\sigma$ in maniera che questi valori siano distinti (linea 23). Supponiamo in fine che v è stato già visitato in passato (quindi $v.\pi$ è GRIGIO o NERO e $v.\sigma$ è già stato determinato) e c'è un arco dal corrente vertice u a v allora l'arco (u, v) forma, insieme al cammino da u a v nell'albero di ricerca, un ciclo C. La parità di C può essere facilmente determinata paragonando $u.\sigma$ e $v.\sigma$ (linea 26): se questi due valori sono uguali allora C ha lunghezza dispari e se sono diversi C ha lunghezza pari.

Supponiamo che la condizione in linea 26 viene soddisfatta da un arco (u, v) e quindi uno di questi vertici, diciamo u viene aggiunto alla coda C. A questi punto la condizione in linea 13 viene soddisfatta è l'algoritmo correttamente ritornerà che il grafo G non è bipartito.

Algorithm 2 DFS modificata per controllare se un grafo non diretto è bipartito o no.

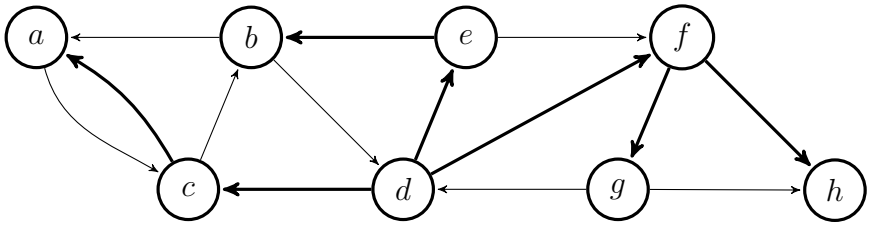
Input: grafo non diretto G.
Output: una bipartizione di G se G è bipartito e `FALSE`.

```
 1: function DFS(G)
 2:     for u ∈ G.V do
 3:         u.c=BIANCO
 4:         u.π=nil
 5:     t = 0
 6:     C ↤ queue, inizialmente vuota
 7:     U ↤ queue, inizialmente vuota
 8:     W ↤ queue, inizialmente vuota
 9:     for u ∈ G.V do
10:         if u.c==BIANCO then
11:             u.σ = 0
12:             DFS-Visit(G,u)
13:             if C not empty then
14:                 return FALSE
15:     return U and W
16: function DFS-Visit(G)
17:     t = t + 1
18:     u.d = t
19:     u.c=GRAY
20:     for v ∈ G.Adj[u] do
21:         if v.c==BIANCO then
22:             v.π = u
23:             v.σ = 1 − u.σ
24:             DFS-Visit(G,v)
25:         else
26:             if u.σ == v.σ then
27:                 C.enqueue(u)
28:                 return
29:     u.c=NERO
30:     t = t + 1
31:     u.f = t
32:     if u.σ = 0 then
33:         U.enqueue(u)
34:     else
35:         W.enqueue(v)
36:     return
```

Finalmente consideriamo il caso in cui la condizione presente nella linea 26 non si verifica mai. Allora ogni vertice u sarà aggiunto ad una sola tra due liste: se $u.\sigma = 0$ allora u viene aggiunto alla lista U (linea 33) e viene aggiunto alla lista W altrimenti (linea 35). Questa bipartizione viene riportata dall'algoritmo principale (linea 15).

Esercizio 3 (I. Salvo)**.** Si consideri il grafo diretto G illustrato nella figura qui sotto e l'albero T formato formato dagli archi evidenziati. L'albero T può essere prodotto da una ricerca in profondità?

- In caso positivo, esibire una rappresentazione di G tramite liste di adiacenza in grado di produrre T e specificare il nodo da cui parte la ricerca e il tipo degli archi ottenuto a seguito della visita.
- In caso negativo, rimpiazzare un arco di T con un altro arco in maniera da ottenere un albero T' con la proprietà che T' possa essere un albero di ricerca per una ricerca in profondità. In tal caso, esibire una rappresentazione di G tramite liste di adiacenza in grado di produrre T' e specificare il nodo da cui parte la visita e il tipo degli archi ottenuto a seguito della visita.



In fine, che succede se si considera lo stesso grafo G dove però gli archi non sono diretti?

Soluzione 3. No, T non può essere ottenuto da una ricerca in profondità, per spiegare il perché è necessario analizzare ogni singolo vertice di G come vertice di partenza.

- a non può essere il vertice di partenza perché altrimenti l'algoritmo dovrebbe visitare c ma (a, c) non fa parte di T;
- b non può essere il vertice di partenza perché altrimenti l'algoritmo dovrebbe visitare uno tra a e d ma sia (b, a) che (b, d) non fa parte di T;

- c non può essere il vertice di partenza perché dopo aver visitato a, l'algoritmo dovrebbe visitare b ma l'arco (c, b) non fa parte di T.

- e non può essere il vertice di partenza perché dopo aver visitato b, l'algoritmo dovrebbe visitare uno tra a e d ma sia (b, a) che (b, d) non fa parte di T;

- g non può essere il vertice di partenza perché altrimenti l'algoritmo dovrebbe visitare uno tra d e h ma sia (g, d) che (g, h) non fanno parte di T;

- il vertice h non avendo archi uscenti è ininfluente per la risposta all'esercizio: se h viene selezionato come vertice di partenza allora esso farà parte di un albero di cui h è l'unico vertice. Se altrimenti h non è il vertice di partenza, h è una foglia del albero di ricerca con g o f come padre. In sintesi, T può essere prodotto da una ricerca in profondità se e solo se T_h può essere prodotto da una ricerca in profondità, dove T_h è il sottografo di G_h ottenuti da T e G rimuovendo, rispettivamente, il vertice h e tutti gli archi ad esso incidenti.

- l'ultimo vertice da analizzare è d. La sequenza $d \to f \to h$ seguita da $f \to g$ è fino a questo punto una legittima sequenza di ricerca in profondità. Adesso l'algoritmo deve visita uno dei vertici adiacenti a d che non sia stato già visitato, cioè o c o e. Se l'algoritmo visita è per primo c, allora il vertice b dovrebbe essere visitato per la prima volta da c usando l'arco (c, b) che però non fa parte di T. Infine se l'algoritmo visita e per primo, allora il vertice a dovrebbe essere visitato per la prima volta da b usando l'arco (b, a) che però non fa parte di T. E' abbastanza immediato anche che scegliere sequenze iniziali diverse da $d \to f \to h$ $f \to g$ producono situazioni simili a quelle dei primi cinque vertici considerati.

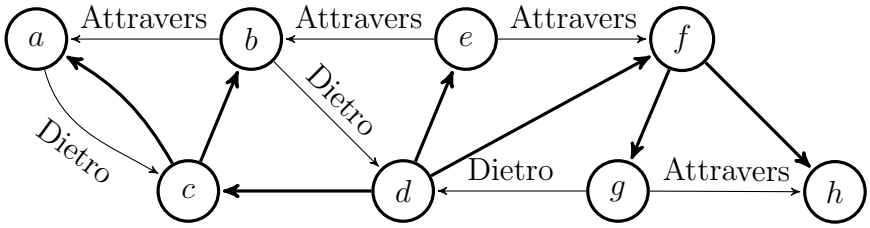
In sintesi, abbiamo mostrato come nessun vertice può essere scelto come vertice iniziale per una ricerca in profondità di G.

Consideriamo ora T', l'albero ottenuto da T rimuovendo l'arco (e, b) e includendo l'arco (c, b). Mostriamo ora che T' può essere ottenuto da una ricerca in profondità. Prima specifichiamo le liste di adiacenza:

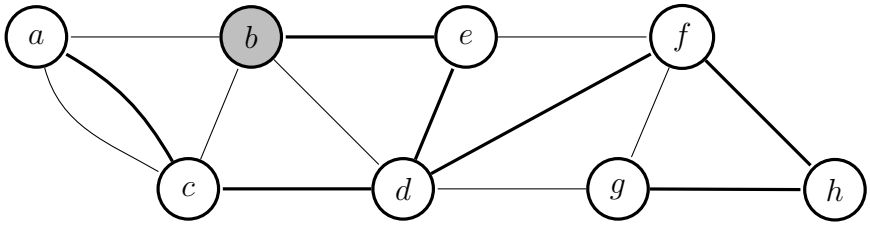
- $a : c$

- $b : a \mapsto d$
- $c : a \mapsto b$
- $d : f \mapsto c \mapsto e$
- $e : b \mapsto f$
- $f : h \mapsto g$
- $g : d \mapsto h$
- $h :$

La figura sotto rappresenta G con l'albero T' evidenziato e gli archi che non sono etichettati con il loro tipo.

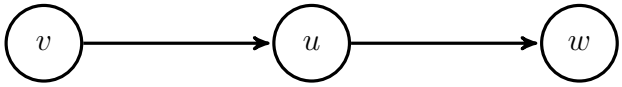


Nel caso G senza orientazione degli archi, la risposta sarebbe comunque negativa ma per motivi differenti. Ci limitiamo ad esibire un albero T' che può essere ottenuto da una ricerca in profondità dove il vertice colorato è quello da dove parte la ricerca.



Esercizio 4 (22.3-11,[1])**.** E' possibile avere un vertice u di un grafo diretto G che finisce in un albero di ricerca in profondità che contiene solo u, anche se G ha almeno un arco entrante ed uno uscente da u? E se si, fornire almeno un esempio non banale di un vertice u con tale proprietà. Che succede se invece G è un grafo non diretto?

Soluzione 4. Dato un grafo G la sua foresta di ricerca dipende fortemente da come vengono selezionati i vertici del grafo nei vari passaggi dell'algoritmo e da come sono ordinate le liste di adiacenza.



Consideriamo il grafo in figura e applichiamo l'algoritmo di ricerca in profondità. Supponiamo che l'algoritmo di ricerca in profondità seleziona come vertice di partenza w. Non avendo alcun arco uscente, w, l'intervallo di visita di w è $i_w = [1, 2]$. Ora, l'algoritmo seleziona un nuovo vertice di partenza e supponiamo che tale vertice sia u. Avendo solo il nodo w come vicino, la cui visita è già terminata, allora $i_u = [3, 4]$. Infine, l'algoritmo seleziona il nuovo e ultimo vertice di partenza v. Avendo solo il nodo u come vicino, la cui visita è già terminata, allora $i_v = [5, 6]$. A questo punto l'algoritmo termina e produce una foresta di ricerca composta da tre vertici isolati. E' utile notare come gli archi (v, u) e (u, w) sono entrami di attraversamento. Questo esempio è una soluzione (minimale) all'esercizio.

Consideriamo ora il caso di grafi non diretti. L'unico esempio che funziona in questo caso è del grafo con un solo nodo u e un solo arco (u, u). Chiaramente è una soluzione ma è "banale". Dimostriamo che non ne esistono altre. Supponiamo che G contiene u e almeno un altro vertice v adiacente ad u. Allora è abbastanza semplice dimostrare che u e v appartengono allo stesso albero della foresta di ricerca in profondità di G per ogni possibile maniera di selezionare i nodi di G e ordinamento delle liste di adiacenza. Infatti, per ogni visita in profondità di un grafo non diretto, non sono presenti archi di attraversamento e quindi due vertici adiacenti appartengono sempre allo stesso albero della ricerca.

Esercizio 5. 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 5. 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 $V(G) = \{v_1, \leq v_n\}$, 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 6 (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 $t < u.d$ e cioè il vertice u deve ancora essere stato visitato per la prima volta allora u è colorato di BIANCO. Se $u.d \leq t \leq u.f$ e cioè il vertice u è ancora in visita allora u è colorato di GRIGIO. Infine, se $t > u.f$ 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 in 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			

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

Soluzione 6. Per il caso di un grafo diretto avremo:

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

References

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