# Esercitazione 1: Depth-First Search

Giacomo Paesani

March 7, 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 aciclico o no; 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 all'indietro.

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 controllare se il grafo diretto in esame è aciclico o no. Supponiamo che durante l'algoritmo viene trovato un arco $(u, v)$ all'indietro (linea 27) allora la variabile *test* viene posta `FALSE` e ritornata subito. Questo vuol dire che la chiamata alla funzione **DFS-Visit** con input $(G, z)$, dove $z$ è l'antenato dell'albero che è anche radice di $u$, ritorna `FALSE` in linea 15 e quindi la funzione **DFS** con input $G$ ritorna subito `FALSE`.

Supponiamo ora che la condizione in linea 27 non viene mai soddisfatta e quindi che non vi sono archi all'indietro. Allora la variabile *test* assume sempre il valore`TRUE` e quindi anche l'output di **DFS**$(G)$ è `TRUE`. $\square$

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

```
global variables
    Color ↤ array di |V| elementi
    Parent ↤ array dei padri
    t ↤ array dei tempi di inizio visita
    T ↤ array dei tempi di fine visita
    test ↤ variabile booleana, inizializzata come TRUE
end global variables
```

**Input:** grafo diretto $G = (V, E)$.

**Output:** `TRUE` se $G$ è aciclico e `FALSE` altrimenti.

```
function DFS(G)
    for u ∈ V do
        Color[u]=BIANCO
    t = 0
    for u ∈ V do
        if Color[u]==BIANCO then
            Parent[u] = u
            test=DFS-Visit(G,u)
            if test=FALSE then
                return FALSE
    return TRUE
function DFS-Visit(G,u)
    t = t + 1
    t[u] = t
    Color[u]=GRAY
    for v ∈ Adj[u] do
        if Color[u]==BIANCO then
            Parent[v] = u
            test=DFS-Visit(G,v)
        if Color[v]==GRIGIO then
            return FALSE
    Color[u]=NERO
    t = t + 1
    T[u] = t
    return test
```

**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 in cui $G$ 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.

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. L'array $Parity$ ha l'obbiettivo di salvare la parità o disparità della lunghezza del cammino dal vertice di partenza della ricerca ad ogni vertice nel albero di ricerca. Per il vertice di partenza della ricerca $z$, $Parity[z]$ è inizializzato come 0 in maniera arbitraria (linea 17). 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 $Parity[v] = 1 - Parity[u]$ in maniera che questi valori siano distinti (linea 29). Supponiamo in fine che $v$ è stato già visitato in passato (quindi $Color[v]$ è GRIGIO o NERO e $Parity[v]$ 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 $Parity[u]$ e $Parity[v]$ (linea 34): 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 34 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 19 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 $G = (V, E)$.
**Output:** una bipartizione di $G$ se $G$ è bipartito e `FALSE`.

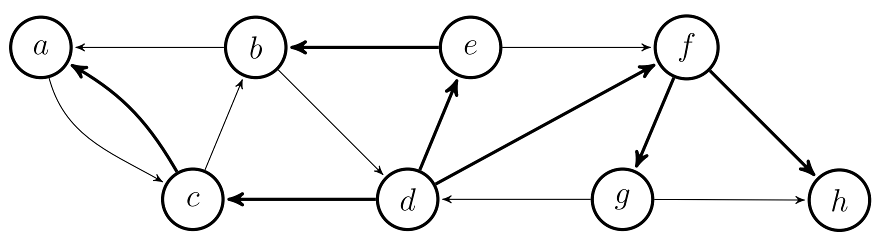
```
global variables
    Color ↤ array di |V| elementi
    Parent ↤ array dei padri
    t ↤ array dei tempi di inizio visita
    T ↤ array dei tempi di fine visita
    U ↤ queue, inizialmente vuota
    W ↤ queue, inizialmente vuota
    test ↤ variabile booleana, inizializzata come TRUE
end global variables
function DFS(G)
    for u ∈ V do
        Color[u]=BIANCO
    t = 0
    for u ∈ V do
        if Color[u]==BIANCO then
            Parent[u] = u
            Parity[u] = 0
            test=DFS-Visit(G,u)
            if test==FALSE then
                return FALSE
    return U and W
function DFS-Visit(G,u)
    t = t + 1
    t[u] = t
    Color[u]=GRAY
    for v ∈ Adj[u] do
        if Color[v]==BIANCO then
            Parent[v] = u
            Parity[v] = 1 − Parity[u]
            test=DFS-Visit(G,u)
            if test=FALSE then
                return FALSE
        else
            if Parity[u] == Parity[v] then
                return FALSE
    Color[u]=NERO
    t = t + 1
    T[u] = t
    if Parity[u] = 0 then
        U.enqueue(u)
    else
        W.enqueue(v)
    return TRUE
```

Finalmente consideriamo il caso in cui la condizione presente nella linea 34 non si verifica mai. Allora ogni vertice $u$ sarà aggiunto ad una sola tra due liste: se $Parity[u] = 0$ allora $u$ viene aggiunto alla lista $U$ (linea 40) e viene aggiunto alla lista $W$ altrimenti (linea 42). Questa bipartizione viene riportata dall'algoritmo principale (linea 21).

**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.

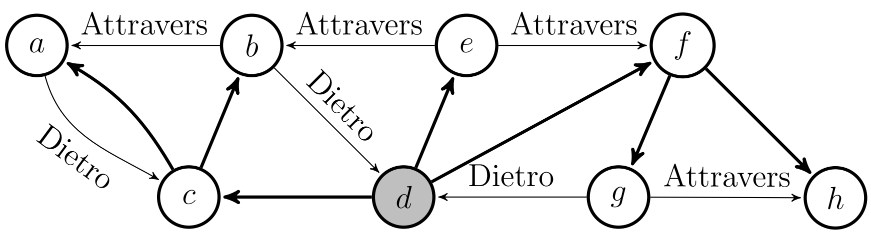
In sintesi, abbiamo mostrato come nessun vertice può essere scelto come vertice iniziale per una ricerca in profondità di $G$.

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:

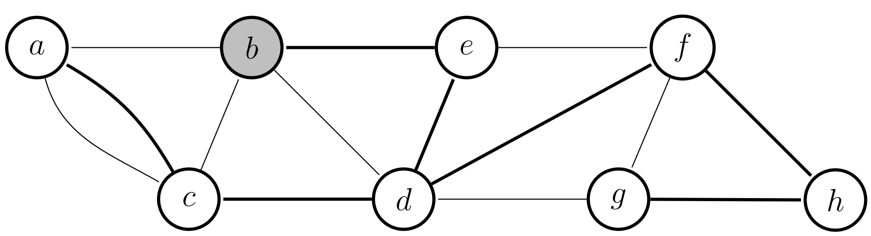
- $a : c$

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

La figura sotto rappresenta $G$ con l'albero $T'$ evidenziato e gli archi che non sono etichettati con il loro tipo. Ora è evidente che avendo come vertice di partenza $d$, l'albero $T'$ è il prodotto della ricerca in fondità su $G$.



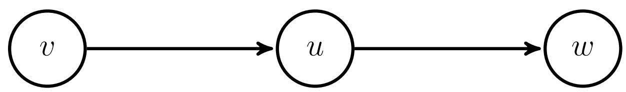

Nel caso $G$ senza orientazione degli archi, la risposta sarebbe comunque negativa ma per motivi differenti. Ci limitiamo ad esibire un albero $T'$ che può essere ottenuto da una ricerca in profondità dove il vertice colorato è quello da dove parte la ricerca.



**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 $u$ ha grado entrante e grado uscente almeno uno?

**Soluzione 4.** Dato un grafo $G$ la classificazione degli archi in base alla visita in profondità 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 solo 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 che non ne esistono altre. Sia $G$ un grafo diretto tale che ci sono due vertici $v$ e $w$ diversi da $u$ tali che $(v, u)$ e $(u, w)$ sono archi di $G$. Supponiamo che $v \neq w$ come in figura (l'altro caso si fa in maniera ancor più semplice).




Se $t[v] < t[u]$ allora $v$ e $u$ appartengono allo stesso albero di ricerca. Se $t[u] < t[w]$ allora $u$ e $w$ appartengono allo stesso albero di ricerca. Ma allora dobbiamo avere $t[w] < t[u] < t[v]$: in questo caso è facile osservare che $u$, $v$ e $w$ appartengono allo stesso albero di ricerca e che gli archi $(v, u)$ e $(u, w)$ sono entrambi di attraversamento.

**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.

# References

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