**Esercizio 2 Cap 3**

Dato un grafo G non direzionato voglio scrivere un algoritmo efficiente che scopre se questo grafo G ha un ciclo, e, nel caso in cui il ciclo esiste, me lo riporta.

Con una visita BFS/DFS, scoprendo un nodo v a partire da u, il nodo v diventa figlio di u nell'albero BFS/DFS, l'arco (u,v) viene inserito. Se esiste un arco (z,v) che non è nell'albero significa che non abbiamo visitato il nodo v a partire da z. Vuol dire che quando noi arriviamo in z e iniziamo ad analizzare gli archi incidenti su z, essendo v già stato scoperto, trovato l'arco (z,v), lo scartiamo. Ciò non esclude però che noi da z possiamo arrivare in v: il percorso che dalla sorgente mi porta in z, con un solo passo mi porta poi in v. Nell'albero è presente l'arco (u,v) e ciò significa che v è raggiungibile dalla radice anche attraverso u. Abbiamo due percorsi alternativi per raggiungere lo stesso nodo a partire dalla radice, e questo è un ciclo.

Il grafo potrebbe non essere connesso ed il ciclo potrebbe essere in una qualsiasi di queste componenti connesse. Dobbiamo eseguire tante BFS/DFS tante quante sono le componenti connesse.

Se dopo l'analisi di tutte le componenti connesse esiste un arco che non fa parte degli alberi BFS/DFS creati, troviamo un arco del grafo in nessuno di questi alberi, c'è un ciclo, altrimenti non c'è.

Algoritmo:

Esegui BFS/DFS fino a che ci sono nodi che non sono stati scoperti, marcando gli archi.

Scandisci gli archi e scopri se l'arco è stato inserito in uno degli alberi.

Comunicare che c'è un ciclo.

Come faccio a dare in output un ciclo? Sono sicuro che c'è un ciclo a partire dalla radice, ma quel che accade è che io potrei avere un antenato comune z prima della radice, ed è quello che dobbiamo trovare. Quindi, il percorso da z a u, e da z a v, più l'arco (u,v) crea un ciclo. Troviamo z mettendoci in u e risalgo verso il padre, fino ad arrivare alla radice. Ogni volta che risalgo il nodo aggiungo il nodo in una lista, mettendolo davanti a quelli precedenti, marcandoli. Faccio la stessa cosa per v. Nel fare questo, però, vado a controllare se i nodi di v sono già stati inseriti nella lista di u, tramite il mark che ho apposto. Non appena incontro un nodo che è stato marcato non inserisco più nodi. Quel nodo è proprio lo z che stavamo cercando, possiamo concatenare le due liste e comunicare il ciclo (facendo in modo che l'antenato compaia due volte O(n+m). Alternativamente posso inserire, sia per u che v tutti gli antenati fino alla radice. Alla fine, siccome li ho messi in testa, posso mettermi nel primo nodo di entrambe le liste e confronto i nodi. Cancello tutti gli elementi che sono uguali. L'ultimo nodo cancellato sarà proprio questo nodo z. Possiamo poi concatenare le due liste aggiungendo z e comunicare il ciclo. O(n+m).

L'analisi che possiamo fare è analoga a quella applicata su AllComponents:

Per creare queste liste impieghiamo tempo lineare sul numero di nodi, O(n).

Il controllo per vedere se ciascun arco, marcato, è all'interno degli alberi, lineare sul numero di archi, O(m).

Se marchiamo e andiamo a cancellare i nodi dalle liste, richiede tempo O(n).

In totale O(n+m).

**Esercizio 3 Cap 3**

Vogliamo modificare l'algoritmo che determina l'ordinamento topologico di un DAG in modo tale che:

se inseriamo in input un grafo G e questo grafo è un DAG

ci restituisce l'ordinamento topologico.

altrimenti se il grafo G non è un DAG

ci restituisce un ciclo del grafo.

Abbia un tempo di esecuzione O(n+m).

Dobbiamo modificare l'algoritmo che determina l'ordinamento topologico.

All'inizio della chiamata ricorsiva ricerca il numero di archi entranti. Se il grafo passato in input è un DAG, l'algoritmo è in grado ad ogni passo di trovare un nodo senza archi entranti e comporre alla fine l'ordinamento topologico. Figuriamo il caso in cui venga inserito in input un grafo che non è un DAG; ad un certo punto ci sarà una chiamata ricorsiva in cui non siamo in grado di trovare un nodo senza archi entranti. Sicuramente in questo sottografo c'è un ciclo. Sia v uno dei nodi di questo grafo. Da v seguo gli archi entranti in v e lo percorro in senso inverso. Dal nodo v arriverò nel nodo u e farò la stessa cosa su un nodo z e così via. Ad un certo punto tornerò in v. Man mano che io percorro i nodi li inserisco in testa ad una lista e nel farlo incontrerò un certo nodo w due volte. Posso rimuovere dalla lista il percorso che va da v a w, w escluso e restituirlo in output.

**Esercizio 7 Cap 3**

Supponiamo di avere un grafo G e che ogni nodo abbia almeno n/2 archi. Voglio dimostrare che il grafo è connesso.

Se il grafo è connesso e lo visito con una BFS/DFS, avrò tutti i discovered dei nodi a true e tutti in un unico albero, quindi applico la visita BFS/DFS su un qualsiasi nodo u del grafo. u viene messo nel livello l\_0, e i nodi adiacenti nel livello l\_1. Siccome u ha grado almeno n/2, il livello 1 conterrà almeno n/2 nodi. Prendiamo un qualsiasi altro nodo v che non si trova nei primi due livelli. v, per ipotesi avrà almeno n/2 archi. Sicuramente non ha alcun arco che lo collega ad u, altrimenti v sarebbe stato nel l\_1. Supponiamo per assurdo che nessuno degli archi di v lo colleghi ad uno dei nodi che si trovano in l\_1.

Devo escludere v per ipotesi, per assurdo devo escluderne n/2, quindi escludo n/2 + 1 nodi. Ne rimangono n/2 - 1. Per ipotesi, però, v deve avere almeno n/2 archi, quindi è impossibile che gli archi di v lo colleghino a nodi che non si trovano in l\_1. v deve essere necessariamente connesso a qualche nodo che si trova in l\_1. Una singola BFS/DFS, quindi, inserisce tutti i nodi del grafo all'interno dell'albero, e l'albero ha 3 livelli, perché i nodi che non verranno inseriti in l\_1 saranno sicuramente inseriti in l\_2.

algoritmo O(m+n) perché devo solo usare la BFS/DFS.

**Esercizio 6 Cap 3**

Supponiamo di avere un grafo non direzionato G e di eseguire su questo grafo sia un BFS che una DFS. Vogliamo dimostrare che se l'albero prodotto dalla BFS è uguale all'albero prodotto dalla DFS allora il grafo di partenza è proprio uguale all'albero prodotto dalle due visite, ossia che il grafo in input era un albero aciclico.

Sia T l'albero costruito dalla BFS e dalla DFS. Supponiamo che G sia diverso da T. Deve quindi esistere almeno un arco (x,y) di G che non appartiene a T. Siccome T è un albero BFS, allora sicuramente l'arco (x,y) congiunge due nodi che si trovano a distanza <= ad 1, quindi che si trovano o nello stesso livello o in livelli adiacenti. T è anche un albero DFS e da ciò sappiamo che se esiste l'arco (x,y), o x è discendente di y nell'albero o viceversa. Trattando lo stesso albero, per la prima considerazione la distanza tra x e y è <= 1 e per la second considerazione ho che x e y sono in relazione antenato-discendente. L'unica possibilità e che x ed y siano quindi in relazione padre-figlio e ciò significa che l'arco (x,y) deve essere nell'albero T. Quindi il grafo G è proprio uguale all'albero T prodotto dalle due visite.