Cicli, Dag e Ordinamenti topologici

Usi meno scontati del DFS

Il DFS, come abbiamo visto, è un algoritmo che in tempo lineare ci permette di trovare tutti i nodi in un grafo.
Possiamo utilizzaralo anche per molte altre funzioni, ma per farlo dobbiamo tenere traccia di due importanti informazioni.
Per ogni nodo terremo traccia del:

- **Momento della prima scoperta** ('previsit');
- **Momento della partenza dal nodo** ('postvisit');

Per tenere traccia di questi momenti possiamo utilizzare una variabile clock, che aumenterà di uno per ogni 'mossa' fatta.

```
procedura visitaDFSRicorsiva(vertice v, albero T):
        marca e visita il vertice v
        pre(v) = clock
        clock = clock + 1
        for each (arco(v,w)) do:
                if(w non è marcato) then:
                        aggiungi l'arco (v,w) all'albero T
                        visitaDFSRicorsica(w,T)
        post(v) = clock
        clock = clock + 1

algoritmo visitaDFS(vertice s) -> albero:
        T <-- Albero vuoto
        visitaDFSRicorsiva(s,T)
        return T
```

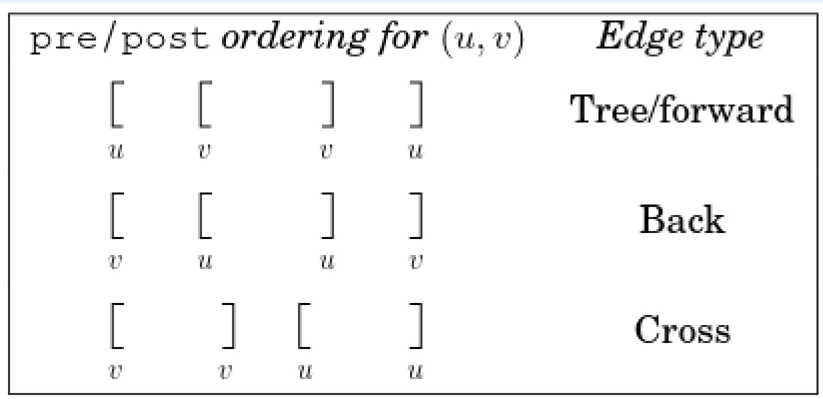
$pre(v)$ -> Tempo in cui viene scoperto v;
$post(v)$ -> Tempo in cui si abbandona v;

Proprietà

Per ogni coppia di nodi u e v, gli intervalli $[pre(u), post(u)]$ e $[pre(v), post(v)]$ o sono disgiunti o l'uno è contenuto nell'altro.

Proprietà

u è antenato di v nell'albero DFS, se $pre(u) < pre(v) < post(v) < post(u)$, condizione che possiamo rappresentare in questo modo:



pre/post *ordering for* (u, v)	*Edge type*
$\underset{u}{[} \; \underset{v}{[} \; \underset{v}{]} \; \underset{u}{]}$	Tree/forward
$\underset{v}{[} \; \underset{u}{[} \; \underset{u}{]} \; \underset{v}{]}$	Back
$\underset{v}{[} \; \underset{v}{]} \; \underset{u}{[} \; \underset{u}{]}$	Cross

Le combinazioni sopra sono tutti i possibili tipi di arco.

Riconoscere la presenza di un ciclo in un grafo diretto

Proprietà

Un grafo G ha un ciclo se e solo se la visita DFS rivela un arco all'indietro.

Si chiamano **DAG (Directed Acyclic Graphs)** tutti quei grafi diretti che non contengono cicli diretti, possono essere usati per risolvere tutti quei problemi in cui abbiamo un ordinamento nello svolgere determinate masioni.

Immagina di avere un grafo diretto, dove i vertici rappresentano delle attività e gli archi rappresentano le dipendenze tra queste attività. Un **ordinamento topologico** è un modo per mettere in sequenza i vertici (le attività) in modo che se un'attività **u** deve essere completata prima di un'altra attività **v** (indicato dall'arco da **u** a **v**), allora **u** viene messa prima di **v** nella sequenza.

Definizione

Un **ordinamento topologico** di un grafo $G = (V, E)$ è una funzione biettiva $\sigma : V \to [1, 2, \dots, n]$ tale che per ogni arco $(u, v) \in E, \sigma(u) < \sigma(v)$.

Da qui nasce il concetto di reti delle dipendenze, cioè grafi nei quali i nodi sono copiti da gestire e dove (u, v) è un arco in cui il nodo u deve essere eseguito prima di v.
Inoltre i nodi possono essere:

- **Pozzi:** Hanno solo archi entranti;
- **Sorgente:** Hanno solo archi uscenti;

Teorema (Dimostrazione sulle slide)

Un grafo G ammette un ordinamento toppologico se e solo se G è un DAG.

L'algoritmo per calcolare un ordinamento topologico è quello della visita ==DFS, ma restituiamo i nodi in ordine decrescente rispetto ai tempi di fine visita ==$post(v)$.

Complessità temporale: $\Theta(n + m)$ se G è rappresentato con liste di adiacenza.
E' corretto per la proprietà numero 1 del foglio.

Componenti fortemente connesse

Una **componente fortemente connessa** di un grafo $G = (V, E)$ è un insieme massimale di vertici $C \subset V$ tale che per ogni coppia di nodi u e v in C, u è raggiungibile da v e v è raggiungibile da u.

Cosa si intende con massimale

Se aggiungiamo un qualsiasi vertice a C la proprietà non è più vera.

Un grafo delle componenti fortemente connesse rimane un DAG.

Come calcolare componenti fortemente connesse da un grafo diretto

Proprietà 1:

Se si esegue la procedura visitaDFSRicorsiva a partire da un nodo u la procedura termina dopo che tutti i nodi raggiungibili da u sono stati visitati.

Idea: Posso quindi eseguire una visita a partire da un nodo di una componente 'pozzo', "eliminare" la componente e ripetere. Ma **come trovo una componente pozzo?**

Proprietà 2:

Se C e C' sono due componenti e c'è un arco da un nodo in C verso uno in C', allora il più grande valore post() in C è maggiore del più alto valore in post() di C'.

Proprietà 3:

Il nodo che riceve da una visita DFS il valore più grande di post() appartiene a una componente sorgente.

```
CompConnesse(grafo G):
        for each (nodo v) do
                imposta v come non marcato
        Comp <- empty
        for each (nodo v in ordine dcrescente di post(v)) do:
                if(v non è marcato):
                        T <-- Albero vuoto
                        visitaDSFRicorsiva(v,T)
                        aggiungi T a Comp
        return Comp
```

Complessità temporale: $\Theta(n+m)$ se G è rappresentato con liste di adiacenza.