Progettazione di Algoritmi - lezione 5

Discussione dell'esercizio [strade critiche]

Possiamo rappresentare la rete viaria con un grafo G non diretto in cui i nodi sono gli incroci e due nodi sono collegati da un arco se c'è una strada che collega i corrispondenti incroci. Per ipotesi G è un grafo connesso. Una strada critica corrisponde a un ponte del grafo G. Per trovare tutti i ponti, un algoritmo molto semplice consiste nell'esaminare ogni arco $\{u, v\}$ considerando il grafo G' che si ottiene rimuovendo l'arco da G e controllare se G' è connesso (se non lo è, l'arco è un ponte, altrimenti non è un ponte). Ma l'algoritmo è molto inefficiente, infatti richiede m visite di grafi che sono quasi uguali a G e quindi ha complessità $O(m(n + m))$. Possiamo fare di meglio?

Possiamo tentare di usare una DFS opportunamente modificata per trovare i ponti. Supponiamo di fare una DFS, a partire da un nodo qualsiasi, del nostro grafo connesso G. Sappiamo che tutti gli archi saranno classificati o come archi dell'albero della DFS o come archi all'indietro. Un arco all'indietro può essere un ponte? No, perché sappiamo che ogni arco all'indietro appartiene ad almeno un ciclo e un ponte non può far parte di cicli. Quindi rimangono solamente gli archi dell'albero. Sia $\{u, v\}$ un arco dell'albero e supponiamo, senza perdita di generalità, che u sia il padre di v. Sia $Tree(v)$ l'insieme dei nodi del sottoalbero della DFS da v. Se c'è un arco all'indietro da un nodo di $Tree(v)$ verso u o un antenato di u, allora l'arco $\{u, v\}$ non è un ponte (perchè c'è un ciclo che contiene l'arco). Viceversa, se non c'è un arco all'indietro da $Tree(v)$ a u o un antenato di u? Supponiamo per assurdo che esista un cammino che collega u e v e che non contiene l'arco $\{u, v\}$. Allora sia z il primo nodo del cammino (percorrendolo da v verso u) che non è in $Tree(v)$. E sia w il predecessore di z, sempre nel cammino, quindi w è in $Tree(v)$. Ne segue che $\{w, z\}$ è un arco da $Tree(v)$ a un nodo fuori di $Tree(v)$ per cui non può essere un arco dell'albero e deve necessariamente essere un arco all'indietro, in contraddizione con l'ipotesi che tali archi non ci sono.

Quindi per determinare se un arco è un ponte basterà controllare che sia un arco dell'albero della DFS e che non ci siano archi all'indietro dal sottoalbero di un estremo dell'arco all'altro estremo o un suo antenato. Per fare questo controllo facciamo in modo che la funzione che esegue la DFS da u ritorni il minimo tra il tempo d'inizio visita di u e il tempo di inizio visita dei nodi antenati di u relativi agli archi all'indietro da $Tree(u)$. Inoltre dobbiamo passargli anche il padre di u per evitare che scambi l'arco tra u e il padre di u per un arco all'indietro.

```
tt: array per i tempi di inizio visita, inizializzato a 0
c <- 0             /* Contatore dei nodi visitati */
P <- lista vuota   /* Lista dei ponti */

DFS_PONTI(G: grafo non diretto, u: nodo, z: nodo, tt: array, c: contatore, P: lista)
    c <- c + 1
    tt[u] = c
    back <- c
    FOR ogni adiacente v di u DO
        IF tt[v] = 0 THEN
            b <- DFS_PONTI(G, v, u, tt, c, P)
            IF b > tt[u] THEN     /* È un arco ponte */
                P.append({u, v})
            back <- min(back, b)
        ELSE IF v <> z THEN
            back <- min(back, tt[v])
    RETURN back
```

La chiamata iniziale sarà `DFS_PONTI(G, u, u, tt, c, P)`. Charamente, la complessità è la stessa di quella della DFS, cioè $O(n + m)$.

Esempio [snodi critici]

Consideriamo una rete ferroviaria formata da lunghe tratte di binari senza interruzioni connesse tra di loro da snodi.

Ogni tratta è percorribile in entrambe le direzioni. Uno snodo può connettere due o più tratte in modo tale che un treno può passare da una tratta a una qualsiasi altra dello snodo. Due snodi A e B sono collegati se un treno partendo da A può arrivare allo snodo B percorrendo gli snodi e le tratte delle rete. Relativamente a due snodi A e B collegati, uno snodo C è detto *critico* se un treno per andare da A a B deve necessariamente passare per C. Data la rete ferroviaria, come possiamo trovare gli eventuali snodi critici per due dati snodi collegati A e B?

È immediato rappresentare il problema tramite un grafo G non diretto i cui nodi sono i snodi della rete e gli archi sono le tratte dei binari. Così un nodo C di G è critico per i nodi A e B (connessi da un cammino) se tutti i possibili cammini che connettono A e B passano per C. Ovvero un nodo D non è critico se e solo se c'è almeno un cammino che collega A e B che non passa per D. Per determinare se D è critico o meno basterà fare una visita a partire da A nel grafo G' in cui il nodo D è stato rimosso, se il nodo B sarà visitato allora D non è critico altrimenti è critico.

Seguendo questo approccio dobbiamo quindi fare n - 2 visite, una per ogni nodo diverso da A e B. Usando una procedura di visita efficiente come una DFS, il nostro algoritmo avrebbe una complessità di $O(n(n + m))$ che è piuttosto elevata. Possiamo trovare un algoritmo più efficiente?

Prima di tutto osserviamo che un nodo critico (per due dati nodi A e B) se rimosso dal grafo lo sconnette (prima c'era almeno un cammino tra A e B e dopo la rimozione non c'è più). In generale, per un grafo non diretto e connesso G, un nodo la cui rimozione sconnette G è detto **punto di articolazione** di G. Quindi, un nodo z è critico per i nodi u e v se z è un punto di articolazione e u e v finiscono in due componenti connesse diverse dopo la rimozione di z.

C'è una certa parentela tra i punti di articolazione e i ponti. Se $\{u, v\}$ è un ponte tale che u ha grado almeno 2, allora u è un punto di articolazione. Però se u è un punto di articolazione, non è detto che qualche arco incidente in u sia un ponte.

Punti di articolazione

Vediamo allora come trovare i punti di articolazione. Facciamo una DFS di un grafo non diretto e connesso G, partendo da un nodo u. Come possiamo riconoscere se u è un punto di articolazione? Chiaramente, una condizione sufficiente affinché non lo sia è che la rimozione di u non sconnetta l'albero di visita (se la rimozione non sconnette l'albero di visita a maggior ragione non sconnette G). È anche una condizione necessaria perché u non sia un punto di articolazione? In altri termini, se la rimozione di u sconnette l'albero di visita, sconnette anche il grafo?

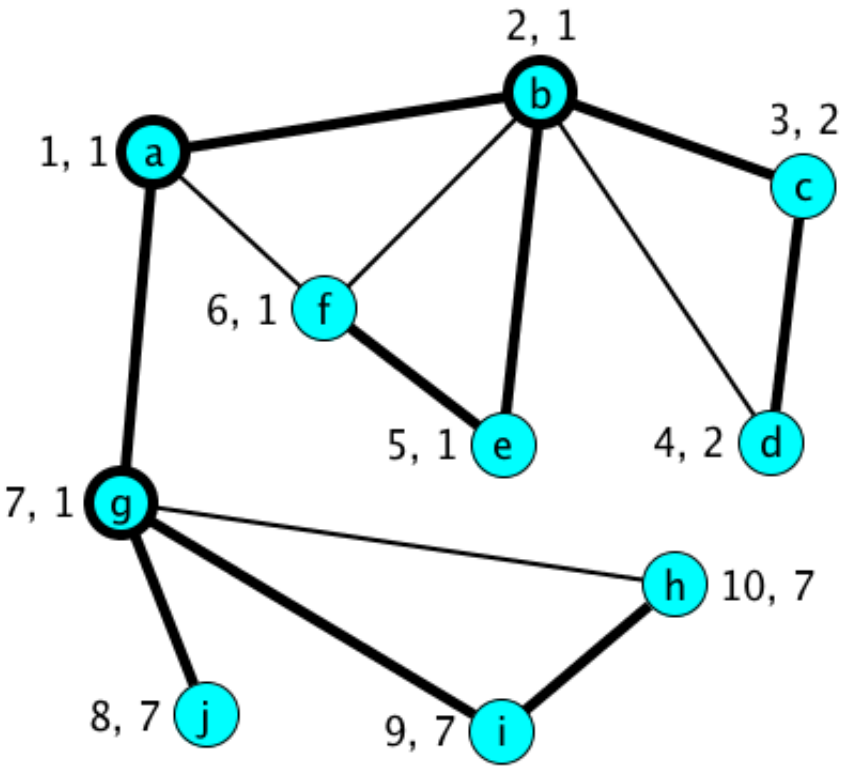
Iniziamo con la radice u dell'albero della DFS. Se la rimozione di u sconnette l'albero, allora u ha almeno due sottoalberi figli. Se eliminiamo u i sottografi relativi a questi sottoalberi saranno connessi solo se ci sono archi tra di essi. Ma non ci possono essere tali archi perchè non sarebbero archi all'indietro. Quindi, se la rimozione di u sconnette l'albero di visita, sconnette anche il grafo. Ne segue che la radice della DFS è un punto di articolazione se e solo se ha almeno due figli. Vediamo ora gli altri nodi. Se un nodo v è un punto di articolazione, la sua rimozione necessariamente sconnette almeno un sottoalbero S della DFS da v. Nel senso che i nodi di S non sono più raggiungibili da u, nel grafo senza v. Questo accade se e solo se non ci sono archi all'indietro da nodi di S a antenati di v. Quindi, un nodo v, diverso dalla radice della DFS, è un punto di articolazione se e solo se esiste un sottoalbero della DFS da v che non ha archi all'indietro verso antenati di v.

Possiamo incorporare queste osservazioni in un algoritmo per trovare i punti di articolazione. Modifichiamo la DFS per mantenere i tempi di inizio visita dei nodi in un array `tt`. Inoltre, per determinare le condizioni circa gli archi all'indietro dei sottoalberi, facciamo sì che la procedura modificata di visita DFS da v ritorni il minimo tempo di inizio visita tra quelli di tutti i nodi toccati durante la DFS da v. Così, un nodo v è un punto di articolazione se e solo se esiste un figlio w di v per cui il valore `b` ritornato dalla visita modificata da w soddisfa `b >= tt[v]`.

```
tt <- array dei tempi inizializzato a 0
c <- 0               /* Contatore dei nodi visitati */
A <- insieme vuoto   /* Insieme dei punti di articolazione */

DFS_ART(G: grafo non diretto, v: nodo, tt: array, c: contatore, A: insieme)
    c <- c + 1
    tt[v] <- c
    back <- c
    children <- 0
    FOR ogni w adiacente di v DO
        IF tt[w] = 0 THEN
            children <- children + 1
            b <- DFS_ART(G, w, tt, c, A)
            IF tt[v] > 1 AND b >= tt[v] THEN
                A.add(v)
            back <- min(back, b)
        ELSE
            back <- min(back, tt[w])
    IF tt[v] = 1 AND children >= 2 THEN
        A.add(v)
    RETURN back
```

La chiamata sarà `DFS_ART(G, u, tt, c, A)`. La figura seguente mostra il risultato dell'esecuzione dell'algoritmo. I nodi marcati sono i punti di articolazione e la coppia di numeri accanto ad ogni nodo *v* riporta il tempo d'inizio visita e il valore ritornato da `DFS_ART` da *v*.



Questo algoritmo è molto più efficiente del primo algoritmo che avevamo considerato. Infatti, la complessità asintotica è uguale a quella di una DFS, cioè O($n + m$). Per trovare i nodi critici relativi a due nodi *u* e *v* è sufficiente iniziare la visita da *u* e valutare un punto di articolazione *w* come nodo critico solo se un suo sottoalbero sconnesso contiene *v*. La definizione dell'algoritmo è lasciata come esercizio.

Un grafo (non diretto) che non ha nodi di articolazione è detto **biconnesso**. Il caso più semplice di grafo biconnesso è un ciclo.

Esercizio [sensi unici]

Le strade di una piccola cittadina sono molto strette e così il sindaco decide di renderle tutte a senso unico. Dopo aver deciso il verso di ogni strada gli viene il dubbio di non averlo fatto nel modo migliore. Quello che vorrebbe è che dati due punti qualsiasi A e B esista sia un percorso lecito da A a B che uno da B a A. Come possiamo aiutarlo a verificare se ciò è vero?