Progettazione di algoritmi

Discussione dell'esercizio [snodi critici]

Data una rete ferroviaria si chiede, dati due snodi collegati A e B, se esistono snodi critici per A e B. Uno snodo C è critico per A e B se un treno per andare da A a B deve necessariamente passare per C. 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. Quindi 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.

Per risolvere il problema con 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. Fra poco vedremo che la DFS ha delle proprietà che possono essere sfruttate per definire un algoritmo efficiente per trovare tutti i punti di articolazione di un grafo. Da questo sarà facile definire anche un algoritmo per i nodi critici.

Proprietà della DFS

Come abbiamo già notato introducendo l'implementazione ricorsiva della DFS, quando un nuovo nodo w è visitato inizia una DFS da w. Questa struttura ricorsiva della DFS ha proprietà molto utili. Per iniziare a studiarle, assegniamo ad ogni nodo v due contatori:

- $t(v)$: il numero di nodi visitati (compreso v) quando inizia la DFS da v;
- $T(v)$: il numero di nodi visitati quando la DFS da v termina.

Si osservi che $t(v) \leq T(v)$ e $T(v) - t(v)$ è il numero di nuovi nodi visitati durante la DFS da v. Entrambi i contatori possono essere interpretati come tempi e infatti $t(v)$ è chiamato *tempo di inizio visita* di v e $T(v)$ *tempo di fine visita* di v. Dati due qualsiasi nodi v e w, o $t(v) < t(w)$ o $t(v) > t(w)$, cioè $t(v)$ e $t(w)$ non possono essere uguali. I nodi visitati possono quindi essere ordinati secondo i tempi di inizio visita: $1 = t(v_1) < t(v_2) < \dots$ dove v_i è l'i-esimo nodo visitato dalla DFS.

La struttura ricorsiva della DFS si evidenzia in modo particolare se consideriamo le relazioni tra gli intervalli $[t(v), T(v)]$. Infatti, per due qualsiasi nodi v e w o i loro intervalli sono disgiunti

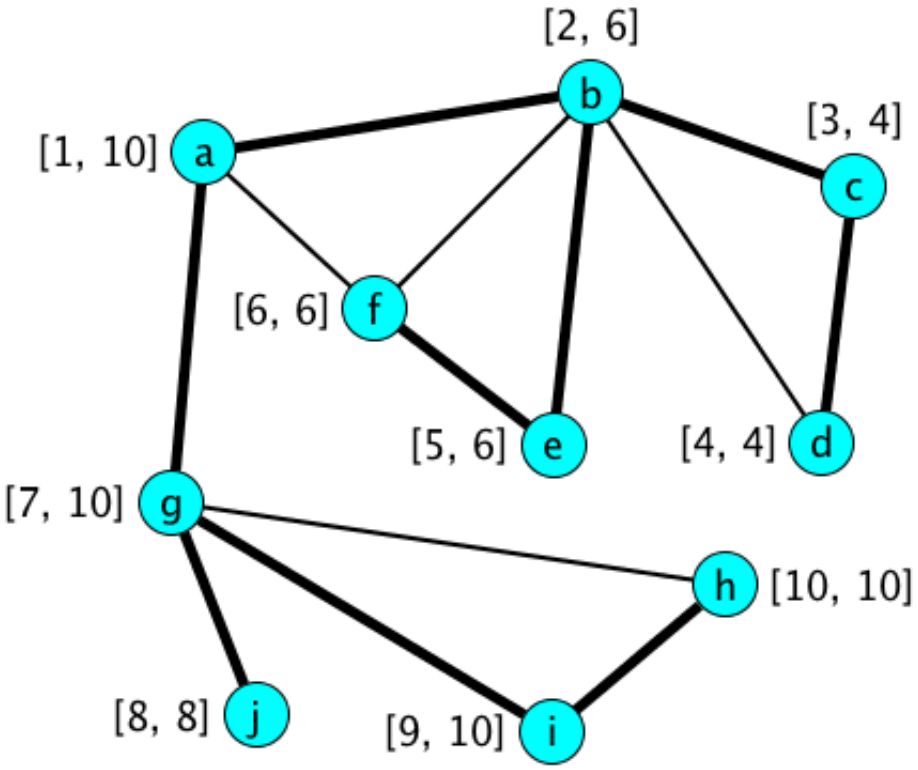
o uno è contenuto nell'altro. Per vederlo supponiamo che $t(v) < t(w)$, se gli intervalli $[t(v), T(v)]$ e $[t(w), T(w)]$ non sono disgiunti, allora $t(v) < t(w) \leq T(v)$. Questo significa che la DFS da w è iniziata quando la DFS da v non è ancora terminata, ne deriva che la DFS da v non può terminare prima che termini la DFS da w. Perciò, $T(w) \leq T(v)$ e l'intervallo di v contiene l'intervallo di w.

Per assegnare i tempi di inizio e fine visita basta modificare leggermente lo pseudo-codice della DFS:

```
TT <- array degli intervalli inizializzati con [0, 0]

DFS_TT(G: grafo, v: nodo, TT: array, c: intero)
    c <- c + 1
    TT[v].t <- c
    FOR ogni w adiacente di v DO
        IF TT[w].t = 0 THEN
            DFS_TT(G, w, TT, c)
    TT[v].T <- c
```

La chiamata iniziale sarà `DFS_TT(G, u, TT, 0)`. Nella figura qui sotto c'è un esempio (gli archi dell'albero di visita sono marcati):



Torniamo ora al problema dei 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?

Per rispondere a questa domanda dobbiamo capire come può essere messo un arco $\{x, y\}$ che non fa parte dell'albero di visita di una DFS. Supponiamo che $t(x) < t(y)$ (l'altro caso è perfettamente simmetrico). Sappiamo che sono possibili solo due casi per gli intervalli di x e y: o sono disgiunti o sono uno contenuto nell'altro. Nel primo caso, si avrebbe $t(x) \leq T(x) < t(y) \leq T(y)$, cioè la visita da x termina prima che inizi la visita da y. Ma questo non è possibile perché l'arco $\{x, y\}$ è attraversato durante la visita da x e quindi y è visitato durante tale visita. Non è perciò possibile che gli intervalli siano disgiunti. Un arco $\{x, y\}$ non appartenente all'albero di visita di una DFS può solamente essere tra un nodo x e un suo discendente y. Siccome l'arco verrà attraversato per la prima volta durante la DFS da y verso x, cioè da un nodo verso un suo antenato, viene detto **arco all'indietro** (in inglese *back edge*). Quindi ogni arco non appartenente all'albero di una DFS è un arco all'indietro.

Adesso possiamo rispondere alla nostra domanda. Se la rimozione di u (che è la radice dell'albero della DFS) 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.

Che cosa possiamo dire per gli altri nodi del grafo? Se un nodo v, diverso dalla radice della DFS, 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.

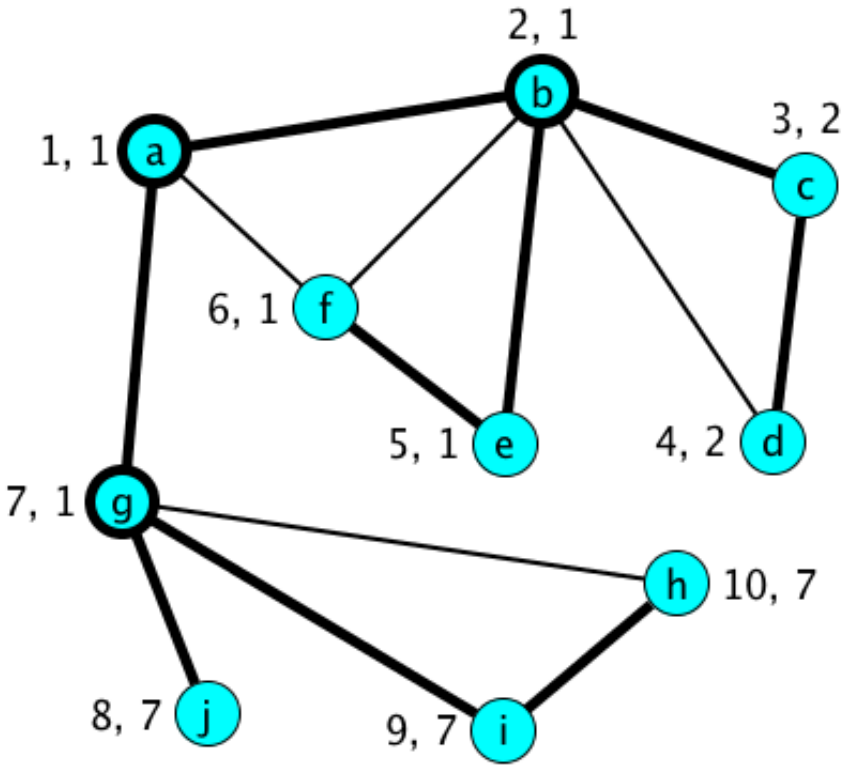
Adesso possiamo incorporare queste osservazioni in un algoritmo per trovare i punti di articolazione. Basterà modificare la DFS in modo da mantenere i tempi di inizio visita (i tempi di fine visita non ci occorrono) dei nodi in un array `tt`, poi un array `A` di booleani per memorizzare i nodi di articolazione (`A[v] = true` significa che `v` e un punto di articolazione). 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 gli adiacenti dei nodi nel sottoalbero della DFS da v. Così, un nodo v è un punto di articolazione se e solo se esiste un adiacente w di v non ancora visitato per cui il valore `b` ritornato dalla visita modificata da w soddisfa `b >= tt[v]`.

```
tt <- array dei tempi inizializzati con 0
A <- array di bool inizializzati con false

DFS_ART(G: grafo, v: nodo, tt: array, c: intero, A: array di bool)
    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 b < back THEN
            back <- b
        IF tt[v] > 1 AND b >= tt[v] THEN
            A[v] <- true
    ELSE IF tt[w] < back THEN
        back <- tt[w]
IF tt[v] = 1 AND children >= 2 THEN
    A[v] <- true
RETURN back
```

La chiamata sarà `DFS_ART(G, u, tt, 0, A)`. La figura seguente mostra il risultato dell'esecuzione dell'algoritmo sullo stesso grafo dell'esempio precedente. 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*.

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

Esercizio [vincoli]

Gli ingegneri che progettano una fabbrica di automobili hanno suddiviso il processo di produzione delle automobili in *n* lavorazioni più semplici. Alcune di queste possono essere effettuate in parallelo (ad es. l'assemblaggio del motore e la saldatura della scocca), ma altre necessitano prima di poter essere iniziate che altre lavorazioni siano state ultimate (ad es. non si può montare la carrozzeria prima di aver preparato il telaio). Per ogni lavorazione *L* gli ingegneri hanno specificato una lista (eventualmente vuota) di lavorazioni che devono necessariamente essere ultimate prima che *L* possa essere iniziata. Vogliamo sapere se le specifiche date sono fattibili, cioè, esiste almeno un ordine di esecuzione delle lavorazioni che rispetta i vincoli dati dalle liste? E poi, se è così, come possiamo trovare un tale ordine?