PROVA PRATICA
SECONDA PROVA IN ITINERE

Traccia 2 – Nodo medio

Sunto

Siano due nodi in un dato grafo non orientato, aciclico e non pesato. Chiameremo distanza, il minor numero di archi necessari a connettere i suddetti e nodo medio, quel nodo da loro equidistante.

Progettare e implementare un algoritmo che, dato un grafo come quello in questione, determini il nodo che risulta essere medio per il maggior numero di coppie di nodi.

Carlo Nicola Cavazzini
Giovanni Maria Cipriano
Riccardo Conti

Ingegneria degli Algoritmi A.A. 2017/2018
Corso di Laurea in Ingegneria Informatica
Università di Roma Tor Vergata

Ingegneria degli Algoritmi A.A. 2017/2018
Corso di Laurea in Ingegneria Informatica
Università di Roma Tor Vergata

Carlo Nicola Cavazzini 0244343
Giovanni Maria Cipriano 0240909
Riccardo Conti 0245745

Sommario

Bibliografia

ALGORITMI E STRUTTURE DATI – Edizione Seconda
Di *C. Demetrescu, I. Finocchi, G.F. Italiano*
Mc-Grow Hill, 2008

DIZIONARIO DELLA LINGUA ITALIANA – Ristampa Seconda
Di *F. Palazzi, G. Folena*
Loescher Editore, 1992

Premessa

Per motivi espositivi, teniamo a sottolineare alcune questioni:

- Ove non altrimenti specificato, considerazioni su grafi G saranno considerazioni su grafi non orientati, aciclici e non pesati;
- Per *grafo non orientato e aciclico* intenderemo che comunque presa una coppia di nodi, esiste unico il cammino che li lega (non ammetteremo l'esistenza di cicli semplici, di conseguenza, nemmeno la possibilità di passare più di una volta su di uno stesso arco);
- Dato un grafo G, siano G_V e G_E rispettivamente l'insieme dei vertici e l'insieme degli archi di G;
- Definizione di *nodi adiacenti* o d'*insieme dei nodi figli* a partire da un *nodo padre* π_x:

$$\forall \pi_x \in G_V : \exists \Phi_{\pi_x} \subset G_V : \Phi_{\pi_x} = \left\{ \varphi_{\pi_x} \middle| \ \varphi_{\pi_x} \in G_V : 1 = dist(\pi_x, \varphi_{\pi_x}) \right\}$$

- Chiameremo "*peso*" di un nodo, in inglese "*weight*", il numero di coppie per cui il suddetto risulta medio.

Il progetto è stato sviluppato per la versione 3.6 di Python. L'ambiente di sviluppo è un sistema basato su x86_64 con processore a 4 core di frequenza 2.8GHz e con 12.9GB di memoria fisica disponibile.

Possiamo fare di meglio, sempre.

Ingegneria degli Algoritmi A.A. 2017/2018
Corso di Laurea in Ingegneria Informatica
Università di Roma Tor Vergata

Carlo Nicola Cavazzini 0244343
Giovanni Maria Cipriano 0240909
Riccardo Conti 0245745

Descrizione dell'algoritmo

A una prima lettura del problema, osserviamo subito che un grafo non orientato e aciclico assume l'aspetto di una foresta di alberi. Questo perché, comunque prendendo una coppia di nodi appartenenti a un grafo totalmente connesso, non orientato e aciclico, esiste, unico, il cammino che li connette.

Concentrandoci invece sulla definizione di nodo medio, viene subito da pensare che ci siano formule per ottenere velocemente il peso di un nodo.

Osserviamo infatti che dalla Combinatoria, in modo pressoché immediato, riusciamo a calcolare il contributo proveniente dai nodi adiacenti a quello di cui calcolare il peso:

$$(i.)\ \forall v_x \in G_V : \exists n \in \mathbb{N} \cup \{0\} : n = \left|\Phi_{v_x}\right| : weight_0(v_x) \stackrel{\text{def}}{=} \frac{n(n-1)}{2} \stackrel{\text{def}}{=} C_{n,2}$$

Ciò segue dal fatto che i nodi adiacenti al nodo di partenza v_x contribuiscono al suo peso in maniera identica al numero di coppie ch'è possibile prendere tra n elementi di classe 2 (vedasi il numero di combinazioni di classe 2 tra n elementi). In particolare, per i nodi aventi un solo *figlio*:

$$(ii.)\ \forall v_x \in G_V : 1 = \left|\Phi_{v_x}\right| : weight_0(v_x) \stackrel{\text{def}}{=} 0$$

Sorgono invece le prime complicazioni quando ci si allontana dal nodo v_x di una distanza superiore a quella di un singolo arco: effettuando una vista in ampiezza del grafo, bisogna assicurarsi che il nodo di partenza rimanga comunque nodo medio.

Infatti, ad ogni k-esimo livello della suddetta vista, non è detto che vi siano sempre un numero esponenziale φ^k nodi che contribuiranno al peso di v_x.

Quindi, dopo aver trovato come gestire il primo livello, osserviamo che i contributi arriveranno dal livello $(k+1)$-esimo che raggiungeremo. Data la complessità delle formule, prenderemo in considerazione solo un rapido esempio, tenendo a mente però, che il ragionamento nasconde un'insidia.

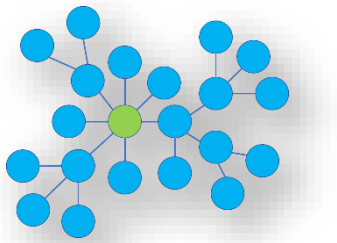


FIGURA 1. *Grafo non orientato. In verde, un nodo medio per 42 coppie di nodi.*

Poniamo di dover calcolare il grado del nodo evidenziato nel grafo in FIGURA 1:

0. Senza dover condurre viste in ampiezza, avremo che i 7 archi contribuiscono in misura di $C_{7,2} = 21$;
1. Alla prima vista, i contributi dalla successiva saranno in numero di $21 = 2 \cdot (3+3) + 3 \cdot (3)$;
2. Alla penultima vista possibile, avremo che gli ultimi contributi saranno $6 = 2 \cdot (3)$;
3. Notiamo infine, che un'ipotetica terza vista sarebbe inutile in quanto non esiste un quarto livello.

A parole, notiamo che dobbiamo combinare tra loro le cardinalità dei nodi sul livello successivo.

Si noti che l'insidia è celata proprio nella condizione per la quale, a un certo punto, mi fermerò: nelle viste in ampiezza non basta verificare che vi siano almeno due nodi aventi ciascuno almeno un *figlio*, ma devo anche assicurarmi che questi due *nodi-padre*, appartengano a rami di discendenza diversi.

Con un po' d'attenzione ci accorgiamo che basta la presenza anche di un solo antenato in comune e nel calcolo dei contributi a un certo k-esimo livello successivo, tutti i nodi contribuiranno come se fossero tutti *figli* di quell'unico antenato (anche se così non fosse!).

Non è un grande passo, quindi, arrivare alla conclusione che l'algoritmo debba calcolare il peso di ogni nodo e quindi restituire quello di peso massimo[1].

[1] I grafi, insiemi non densi, potrebbero avere anche più di un nodo con peso massimo (e.g. si provi a calcolare il peso del nodo a destra di quello evidenziato in FIGURA 1).

Carlo Nicola Cavazzini 0244343
Giovanni Maria Cipriano 0240909
Riccardo Conti 0245745

Analisi del tempo di esecuzione teorico e sperimentale

Analizziamo quindi cosa comporta, dato un grafo, calcolare, per ogni nodo, il relativo peso.

Come prima cosa, non comportando significativi rallentamenti, come scelta implementativa decidiamo di restituire, in una lista, tutti i nodi che hanno lo stesso massimo peso.

Siano:

- $n \coloneqq |G_V|$ il numero di nodi del grafo G in input;
- $gM \coloneqq getMiddles$ l'acronimo della funzione principale;
- $gNOC \coloneqq getNbrOfCouples$ l'acronimo della funzione "secondaria" che calcola "solo" il peso di un nodo.

Appare chiaro che sia cruciale analizzare approfonditamente $gNOC$, in quanto sarà reiterata su ogni nodo presente nel grafo.

Infatti, dato un nodo che chiameremo "radice", $gNOC$ dovrà effettuare una vista *B.F.S.* del grafo. Quest'operazione reiterata su ogni nodo ci porterebbe intuitivamente già a tempi $O(n^2)$. Come se non bastasse, durante ogni relativa discesa *B.F.S.*, bisogna anche calcolare il contributo proveniente dal livello raggiunto. Infine, completata la suddetta vista, dovremo sommare i contributi da ogni livello.

Bisogna notare che perché una coppia di nodi sullo stesso livello della *B.F.S.* abbia come nodo medio la radice è condizione necessaria e sufficiente[2] che i suddetti due nodi abbiano come unico antenato comune la radice e nessun altro nodo ad eccezione di essa.

A un prima più precisa lettura, per ogni nodo di cui calcolare $gNOC$, quest'ultima dovrà:

1. Calcolare le coppie tra i nodi adiacenti alla radice data (vedasi la formula delle combinazioni statistiche), $O(1)$;
2. Generare tante liste quanti sono i nodi adiacenti alla radice (che chiameremo "primi figli"), $O(1)$;
3. Effettuare la *B.F.S.* calcolando, ad ogni livello, i contributi provenienti dal livello successivo, $O(n)$;
    a. Riempire le suddette liste con i nodi del livello successivo (in modo tale che per ogni "primo figlio", la lista corrispondente contenga solo i relativi discendenti), $O(1)$;
    b. Moltiplicare, a coppie, tutte le lunghezze delle liste di discendenti, $O(1)$.
4. Sommare i contributi di ogni livello, $O(1)$.

$$T_{gNOC}(n) = O(n) \Rightarrow T_{gM} = O(n^2)$$

Non è detto, dobbiamo analizzare meglio le varie operazioni.

Quest'insieme di liste contenente i nodi di un livello dell'albero, sarà utilizzato come iterabile da due cicli **for** annidati: il primo scorrerà sul numero di liste totali, mentre quello più interno avrà indici che scorreranno sempre sul numero di liste totali, ma a partire dal successivo indice a cui è giunto il ciclo più esterno, e.g. in codice Python:

1. $\boldsymbol{for}\ i\ in\ range(len(listOfLists) - 1)$:
2. $\quad support = len(listOfLists[i])$
3. $\quad \boldsymbol{for}\ j\ in\ range\big(i + 1, len(listOfLists)\big)$:
4. $\quad\quad couplesFromGeneration\ +\!= support \cdot len(listOfLists[j])$

Si noti come quanti più nodi adiacenti avrà la radice, tante più moltiplicazioni dovremo eseguire ad ogni livello. Ponendo a_0 come il numero di figli adiacenti alla radice, avremo $O(a_0^2)$ moltiplicazioni per ogni possibile radice.

[2] Per i nostri grafi, gli alberi generati dalle *B.F.S.* sono un alberi dei cammini minimi.

Passando al cuore del problema, la *B.F.S.*, abbiamo un ciclo ***while*** che itera sui livelli d'ampiezza.

Visita ogni nodo una e una sola volta (cioè ne estrae la lista dei figli) ed ottenuta la lista con tutta la successiva generazione di nodi esegue al più a_x^2 moltiplicazioni per calcolare il contributo dal livello successivo a quello raggiunto (a_x è il numero di nodi presenti al livello $(x+1)$-esimo).

Sia d la distanza fra la radice e il suo nodo più lontano, si ha quindi che eseguiremo al più d cicli ***while***:

$$T(d, a_x) = d \cdot a_x^2$$

Una stima più coerente per gM diventa:

$$T(n, d, a_x) = n \cdot d_{max} \cdot a_x^2$$

La funzione tiene conto di tre parametri e non solamente del numero di nodi, poiché ad incidere pesantemente sui tempi non è solo il loro numero, ma anche la conformazione del grafo.

Infatti, come si può evincere dai grafici, per grafi molto densi (cioè in cui ogni nodo ha molti figli) all'aumentare dei nodi il tempo per gM aumenta in maniera lineare.

Invece, per grafi sparsi in cui cioè ogni nodo è connesso a pochi nodi, il tempo per trovare il nodo con peso massimo aumenta in maniera quadratica all'aumentare dei nodi.

Questo comportamento è dovuto al fatto che generalmente più saranno densi i grafi, e minore sarà d, e.g.:

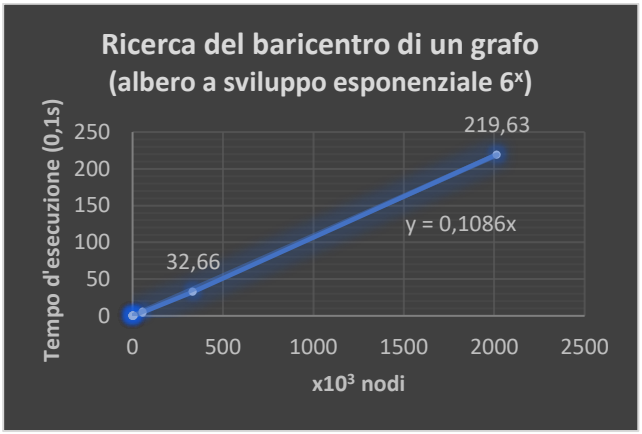
> Costruiamo un grafo a partire da un nodo centrale avente tre nodi adiacenti. Ripetendo per dodici volte l'operazione di attaccare a ogni foglia un numero di tre nuove foglie, avremo un grafo con $3^0 + 3^1 + 3^2 + \cdots + 3^{13} = 797.161$ nodi e con $d = 26$ (i due nodi più distanti saranno due foglie che hanno come unico antenato comune la radice): nel caso peggiore, eseguiremo 24 iterazioni per il ***while*** mentre $k^2 = 9$.
>
> Provando invece a costruire un grafo composto da 700.000 nodi in linea, certo $k^2 = 1$, ma d sarà nel peggiore dei casi 700.000, portando la funzione a spendere molto tempo all'interno del ciclo ***while***.

Da notare alcuni piccoli accorgimenti messi in atto per risparmiare ulteriore tempo:

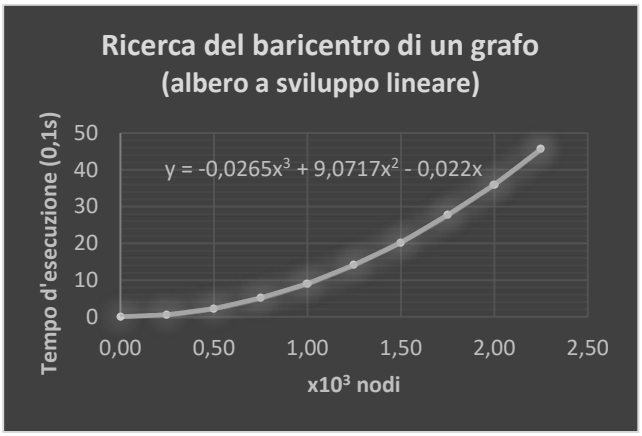
- Se un nodo è foglia, non sarà effettuata la *B.F.S.* poiché non potrà essere mai medio di un percorso;
- Se durante la *B.F.S.*, il numero di coppie di un dato livello, dovesse essere uguale a zero, la *B.F.S.* non proseguirebbe oltre poiché i livelli inferiori continuerebbero ad avere contributo zero.

Carlo Nicola Cavazzini 0244343
Giovanni Maria Cipriano 0240909
Riccardo Conti 0245745

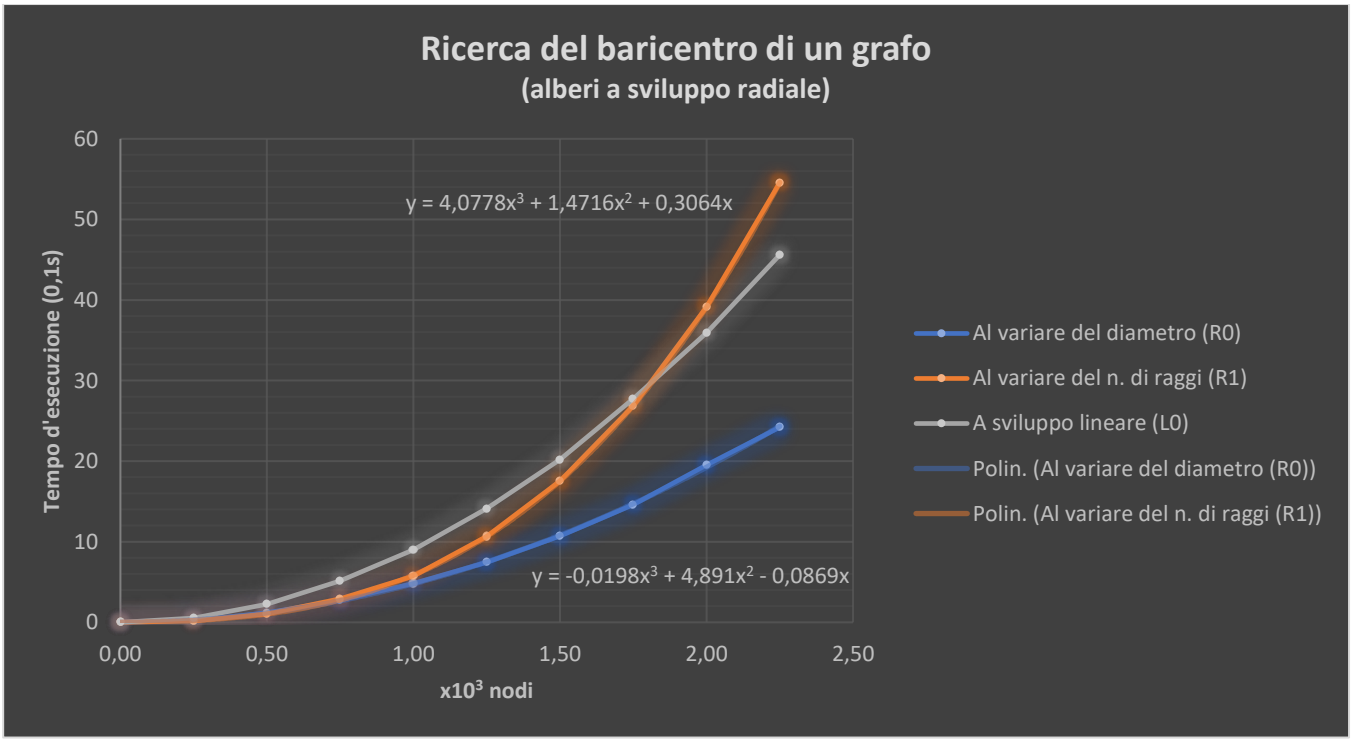
Grafici e tabelle



E0 Tempo d'esecuzione (0,1s)	Migliaia di nodi
0,00	0
0,00	0,001
0,00	0,043
0,00	0,259
0,16	1,555
0,63	9,331
4,84	55,987
32,66	335,923
219,63	2015,539



L0 Tempo d'esecuzione (0,1s)	Migliaia di nodi
0,00	0
0,54	0,25
2,23	0,50
5,13	0,75
9,01	1,00
14,10	1,25
20,19	1,50
27,76	1,75
35,92	2,00
45,60	2,25



Segue la tabella per il soprastante grafico.

Carlo Nicola Cavazzini 0244343
Giovanni Maria Cipriano 0240909
Riccardo Conti 0245745

R0 Tempo d'esecuzione (0,1s)	R1 Tempo d'esecuzione (0,1s)	Migliaia di nodi
0,00	0,00	0
0,21	0,16	0,25
1,18	1,04	0,50
2,77	2,92	0,75
4,84	5,73	1,00
7,43	10,67	1,25
10,71	17,52	1,50
14,60	26,89	1,75
19,54	39,17	2,00
24,21	54,57	2,25

La tabella per l'ultimo grafico della pagina precedente. Si noti come la variazione del diametro dei grafi a sviluppo radiale porti l'algoritmo a comportarsi similmente al caso di grafi a sviluppo lineare.

Extra

Sia questa la sezione dedicata a una trattazione discorsiva e meno rigorosa, alla sperimentazione.

Equazioni per grafi notevoli

Seguono alcuni studi sulla crescita del numero di visite totali necessarie in funzione del numero di nodi.

Grafi a sviluppo lineare degli n nodi

$$T(n) = \left\lfloor \frac{n}{2} \right\rfloor + 2 \cdot \sum_{k=0}^{\lfloor n/2 \rfloor - 1} k = \left\lfloor \frac{n}{2} \right\rfloor + 2 \cdot \frac{\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right) \cdot \left(\left\lfloor \frac{n}{2} \right\rfloor - 1 + 1\right)}{2} \approx O(n^2)$$

Mentre il numero di nodi crescerà linearmente, il numero delle visite crescerà in modo quadratico.

Grafi a sviluppo radiale degli n nodi distribuiti su ρ filamenti di λ nodi ciascuno

Escludiamo il nodo centrale che richiederà la visita di tutti i nodi e anche tutti quei nodi lontani dalla radice più di $\left\lfloor \frac{\lambda}{2} \right\rfloor =: l$.

$$T(n) = \rho \cdot \sum_{k=l+1}^{2l} \big(l + k + k \cdot (\rho - 1)\big) = \rho \cdot \sum_{k=l+1}^{2l} k\big(l + (\rho - 1)\big) =$$

$$= \rho \cdot \big(l + (\rho - 1)\big) \cdot \sum_{k=l+1}^{2l} k = \big(\rho l + \rho(\rho - 1)\big) \cdot \left[\left(\frac{2l \cdot (2l + 1)}{2}\right) - \left(\frac{(l + 1) \cdot (l + 2)}{2}\right)\right] \approx$$

$$= O(\rho^2 + \rho l - \rho) \cdot O(l^2 - 2l - 3) \approx O(\rho^2 \cdot \lambda^2) \approx O(n^2)$$

Mentre il numero di nodi crescerà linearmente ($n = \rho \cdot \lambda + 1$), il numero di visite crescerà in modo quadratico ($\rho^2 \cdot \lambda^2$).

Grafi a sviluppo esponenziale degli n nodi distribuiti su l livelli di b^k nodi ciascuno (dove $k \in [0, l] \subset \mathbb{Z}$)

Anche escludendo il nodo centrale nonché tutti i nodi foglia, intuitivamente avremo che, aggiungendo nuovi livelli, le visite cresceranno in modo asintotico al numero dei nodi ($n = \sum_{k=0}^{l} b^k$).

Possiamo fare di meglio? La sfida sarebbe ricondursi al caso della ricerca del baricentro di un corpo[3].

[3] Purtroppo, non è immediato passare da un grafo, al corrispettivo corpo geometrico di densità uniforme.