Relazione 2° Progetto in Itinere – IA 2017/2018

Studenti

- Foderaro Salvatore
- Menichelli Alberto
- Di Blasi Giorgia

Problema

Sia dato un grafo non orientato aciclico e non pesato G. Data una coppia di nodi (n1,n2) in G , la distanza dist(n1,n2) é il minor numero di archi necessari a connettere n1. Un nodo m é medio di n1 ed n2 se e solo se é equidistante da n1 e n2 , ovvero se dist(n1,m) = dist(m,n2). *Progettare e implementare un algoritmo che, dato un grafo non orientato aciclico e non pesato G, determini il nodo m* che risulta essere medio per il maggior numero di coppie di nodi*.

Descrizione algoritmo

L'idea alla base dell'algoritmo è quella di considerare, in ogni sotto-grafo connesso, i percorsi più lunghi tra due foglie e la lista dei nodi che vi appartengono. Per ottenere il nodo medio per il maggior numero di volte, per ogni percorso basta considerare gli elementi che stanno esattamente a metà della lista. Nel caso in cui la lunghezza del percorso sia dispari (figura 1), l'elemento a metà è solamente uno e rappresenta esattamente l'Id del nodo che sto cercando.

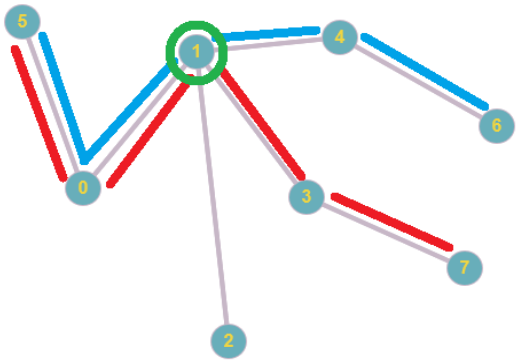


Figura 1

Nel caso in cui invece la lunghezza del percorso dovesse essere pari (figura 2), gli elementi che si trovano a metà sono due. Per sapere quali dei due risulta medio per il maggior numero di volte, mi basta eliminare l'arco che collega i due nodi ed eseguire due visite generiche separate, andando in questo modo a contare il numero di nodi connessi ai due nodi considerati. Risulterà medio per il maggior numero di volte il nodo con più nodi connessi, mentre risulteranno tutti e due medi nel caso in cui questa cifra dovesse essere uguale.

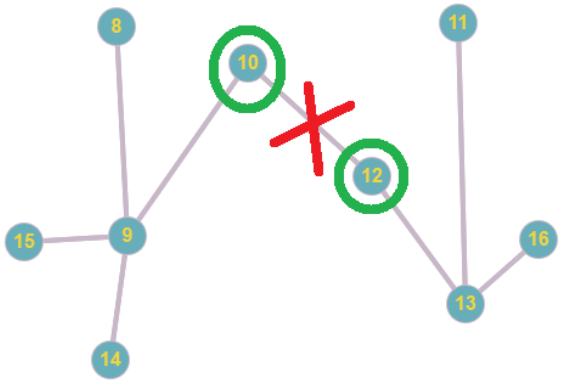


Figura 2

Nell'eventualità che il numero dei nodi medi per il maggior numero di volte all'interno del sotto-grafo sia maggiore di uno, come struttura di appoggio utilizzo una lista, dove vado a memorizzare la lista dei nodi che risultano medi per il maggior numero di volte, ed il numero esatto di volte che essi risultano medi (lunghezza del percorso diviso due, nel caso in cui la lunghezza del percorso sia dispari; numero di nodi connessi, nel caso in cui il percorso sia pari).

Per ogni percorso massimo, memorizzo la lista dei nodi che sono risultati massimi ed il numero di volte che lo sono stati. In un'altra lista memorizzo tutti i nodi che sono risultati massimi. L'idea è quella di considerare ogni nodo che è risultato massimo, impostare il suo contatore sul numero di volte che è stato massimo a 0, e per ogni percorso controllare se il nodo risulta presente. Se ci sta

in quel percorso, sommo al suo contatore il numero di volte che i nodi nella lista sono risultati medi e, terminata l'iterazione, confronto il suo contatore con quello del nodo massimo. Se maggiore, sostituisco le informazioni del nodo massimo, se uguale invece lo aggiungo alla lista dei nodi massimi.

Il punto di forza dell'algoritmo è il non dover considerare ogni nodo, ma bensì solamente le singole foglie per trovare il percorso più lungo tra due di loro. Inoltre, una volta trovata la foglia "più profonda", sfruttando la proprietà degli alberi, per ottenere la lista dei nodi appartenenti al percorso, mi basterà risalire l'albero accedendo continuamente a padre del nodo successivo, fin quando questo non avrà valore nullo.

Trattandosi di un grafo non connesso, è stata eseguita un'ottimizzazione della funzione **getAdj.** Se infatti la funzione originaria restituiva la lista di tutti i nodi adiacenti ad un nodo dato, **getAdjModified** controlla se il nodo ha almeno un nodo adiacente; in caso positivo, apparterrà sicuramente ad un sotto-grafo connesso. Inoltre per ogni foglia profonda trovata, viene effettuato il controllo se la foglia si trova già nella lista dei nodi da considerare, ed in caso contrario la inserisce.

Strutture dati utilizzate

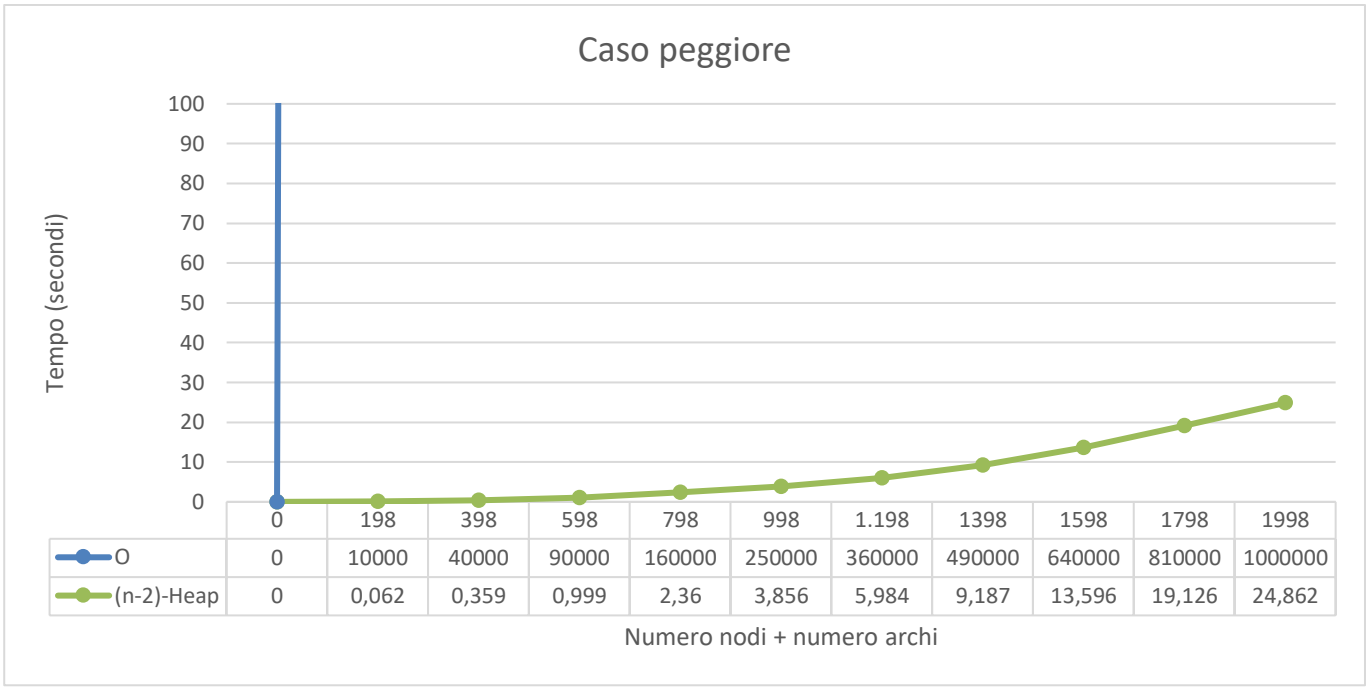
- **TreeArrayList (**Aggiunta del campo **Distanza** alla classe **Node)**
- **GraphAdjacentyLists**

Analisi tempo teorico

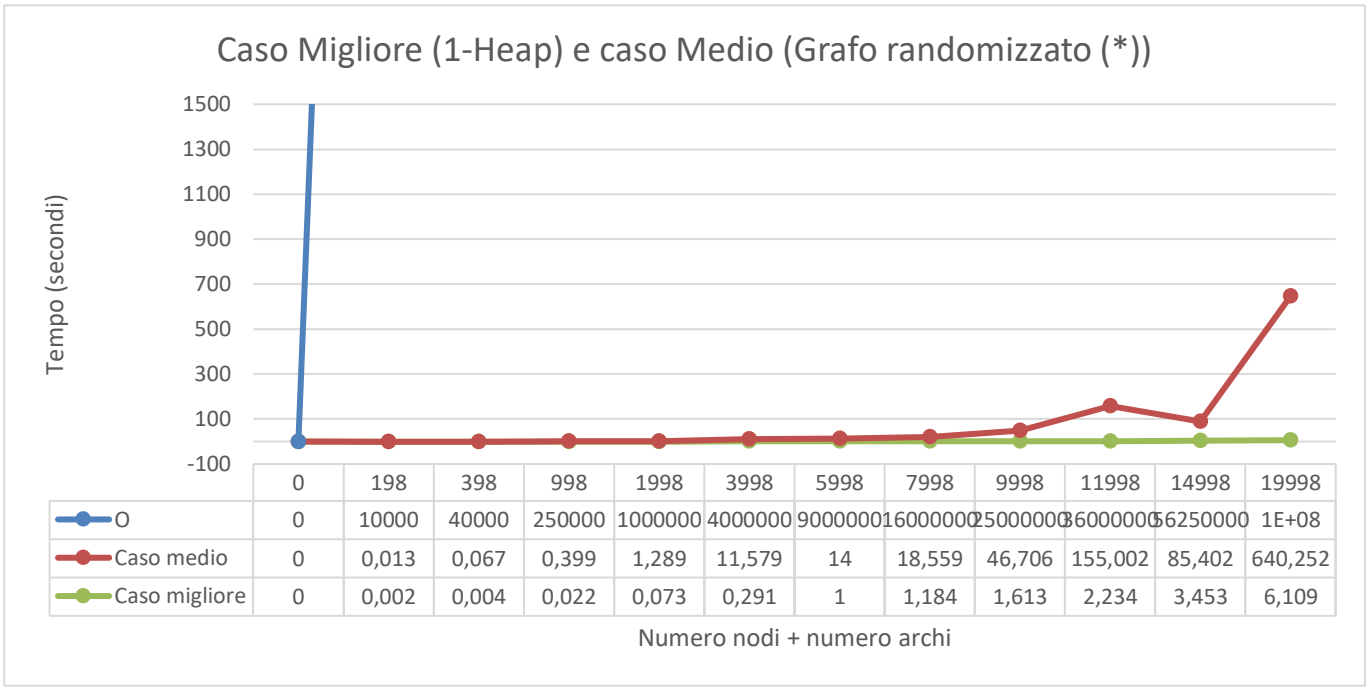
- leafDistance: $\boldsymbol{O(m+n)}$ | Eseguo una visita generica a partire dalla foglia, che viene considerata come radice dell'albero
- mediumNode: $\boldsymbol{O(m+n) + \left(numero_{foglie} * leafDistance\right) = O(m+n) + (n-2) * O(m+n) = O(n^2)}$ | Eseguo una visita generica a partire da un nodo qualsiasi del grafo e, nel caso peggiore, il numero delle foglie è uguale ad $(n-2)$
- calculateSubNode: $\boldsymbol{O(m+n)}$ | Nel caso in cui la lunghezza del percorso dovesse essere pari, a partire da ognuno dei due candidati a nodo medio, eseguo una visita generica per ottenere il numero di elementi a loro connessi
- backToFather: $\boldsymbol{O(1) * calculateSubNode = O(m+n)}$ | Nel caso peggiore, la lunghezza del percorso è pari
- controlloFunzione: $\boldsymbol{O(n) + \left(numero_{foglie} * backToFather\right) + mediumNode = O(n) + O(n-2) * O(m+n) + O(n^2) = O(n^2))}$ | Controllo tutti i nodi per eliminare quelli non connessi a nessun sotto-grafo, e nel caso peggiore il numero di foglie è $(n-2)$

Analisi tempo sperimentale

L'analisi sperimentale conferma l'analisi teorica e mostra, inoltre, come la complessità temporale dell'algoritmo sia strettamente correlata col il numero di foglie appartenenti al grafo. Il caso migliore si presenta con un grafo assimilabile ad un 1-Heap + 1 nodo disconnesso, in quanto è presente una singola foglia. Il caso peggiore, come è facile immaginare, si ha nel caso di un (n-2)-Heap, dove n rappresenta il numero dei nodi appartenenti al grafo.



	0	198	398	598	798	998	1.198	1398	1598	1798	1998
O	0	10000	40000	90000	160000	250000	360000	490000	640000	810000	1000000
(n-2)-Heap	0	0,062	0,359	0,999	2,36	3,856	5,984	9,187	13,596	19,126	24,862



	0	198	398	998	1998	3998	5998	7998	9998	11998	14998	19998
O	0	10000	40000	250000	1000000	4000000	9000000	16000000	25000000	36000000	56250000	1E+08
Caso medio	0	0,013	0,067	0,399	1,289	11,579	14	18,559	46,706	155,002	85,402	640,252
Caso migliore	0	0,002	0,004	0,022	0,073	0,291	1	1,184	1,613	2,234	3,453	6,109

(*) Il grafo è stato generato in modo random utilizzando la funzione **createRandomGraph** disponibile nel file **demoAlgoritmo.py**