

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 $\text{dist}(n1, n2)$ è il minor numero di archi necessari a connettere $n1$. Un nodo m è medio di $n1$ ed $n2$ se e solo se è equidistante da $n1$ e $n2$, ovvero se $\text{dist}(n1, m) = \text{dist}(m, n2)$. *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, il percorso più lungo tra due foglie e la lista dei nodi che vi appartengono. Per ottenere il nodo medio per il maggior numero di volte all'interno del percorso, basta considerare gli elementi che stanno esattamente a metà della lista. Nel caso in cui la lunghezza del percorso sia dispari, l'elemento a metà è solamente uno e rappresenta esattamente l'id del nodo che sto cercando.

Nel caso in cui invece la lunghezza del percorso dovesse essere pari, 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.

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).

Inizializzando le informazioni del nodo massimo dell'intero grafo a 0, per ogni percorso più lungo di ogni sotto-grafo connesso mi basterà confrontarle con il valore relativo al numero di volte che il nodo (o i nodi) è risultato medio all'interno del sotto-grafo, ed in caso impostare lui come nuovo nodo massimo.

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.

Strutture dati utilizzate

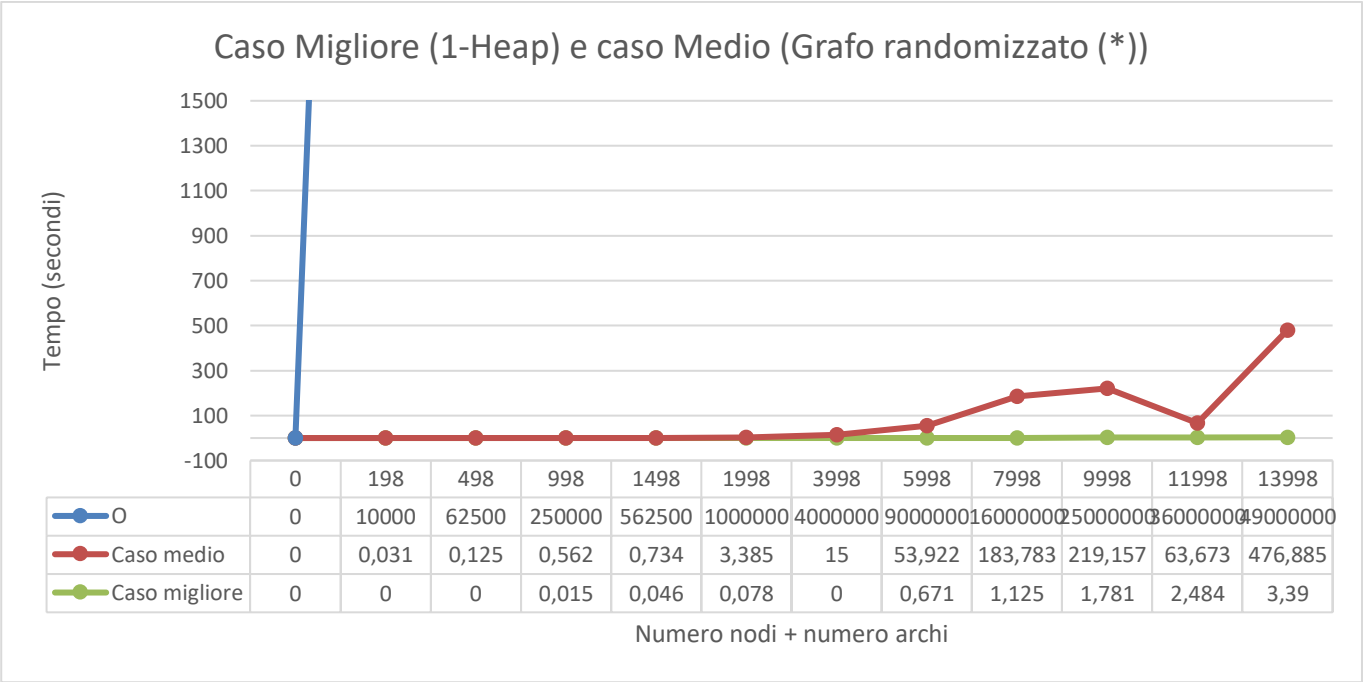
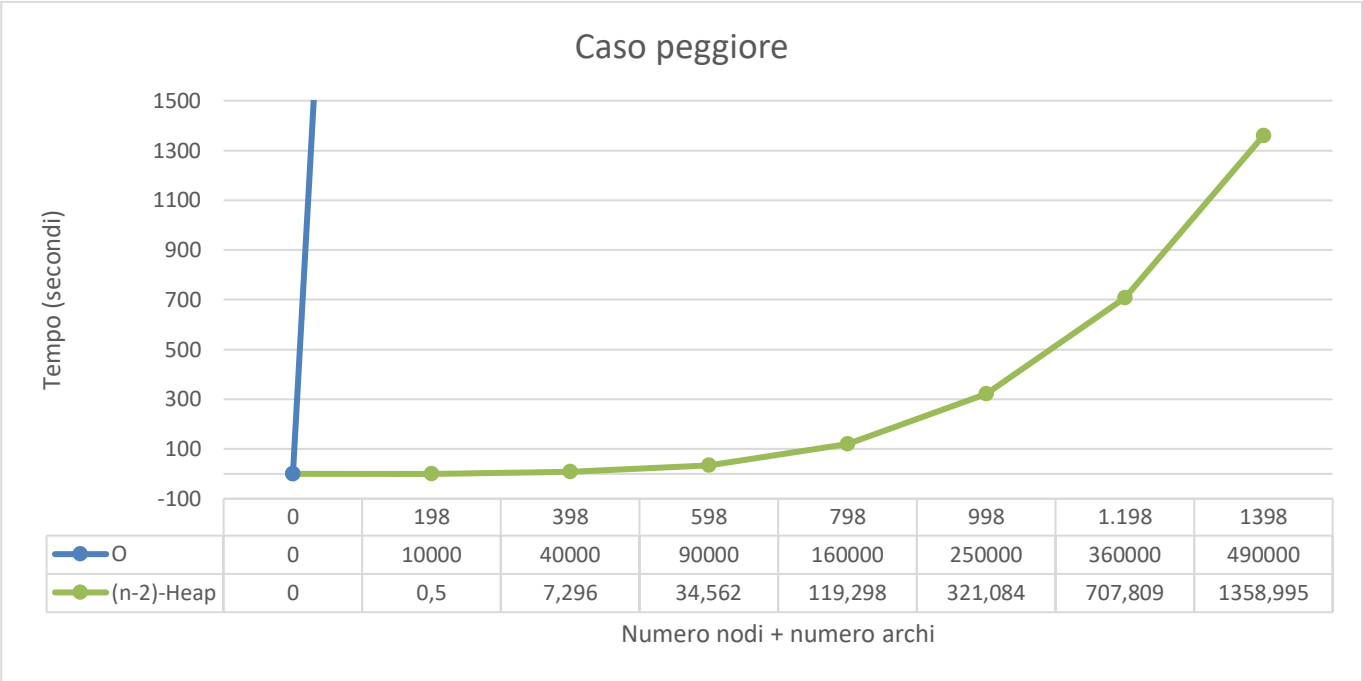
- **TreeArrayList** (Aggiunta del campo **Distanza** alla classe **Node**)
- **GraphAdjacencyLists**

Analisi tempo teorico

- leafDistance: $O(m + n)$ | Eseguo una visita generica a partire dalla foglia, che viene considerata come radice dell'albero
- mediumNode: $O(m + n) + (\text{numero}_{foglie} * \text{leafDistance}) = O(m + n) + (n - 2) * O(m + n) = O(n^2)$ | 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: $O(m + n)$ | 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: $O(1) * \text{calculateSubNode} = O(m + n)$ | Nel caso peggiore, la lunghezza del percorso è pari
- controlloFunzione: $O(n) + (\text{numero}_{foglie} * \text{backToFather}) + \text{mediumNode} = O(n) + O(n - 2) * O(m + n) + O(n^2) = O(n^2)$ | 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.



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