**Relazione 2° Progetto in Itinere – IA 2017/2018**

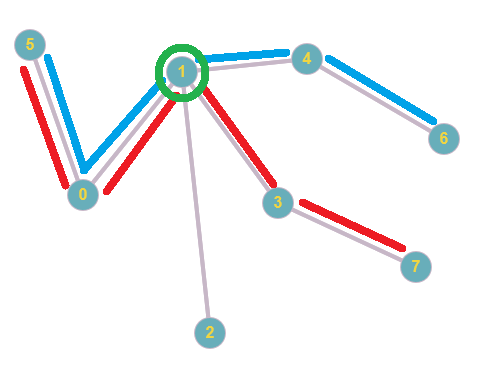
**Studenti**

* 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) 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 ed n2, ovvero se dist(n1,m)=dist(m,n2). *Progettare ed 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.*

(La non aciclicità del grafo è stata interpretata nel modo seguente: inserisco un arco tra due nodi solamente se la testa dell’arco non è già presente nella lista dei nodi considerati come testa. In questo modo, ogni nodo ha al più un arco in entrata, non considerando l’arco inverso per soddisfare la proprietà del non orientato. Inoltre, non essendoci indicazione a riguardo, il grafo è considerato come non connesso).

**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 ed è rappresentato esattamente l’Id del nodo cercato.

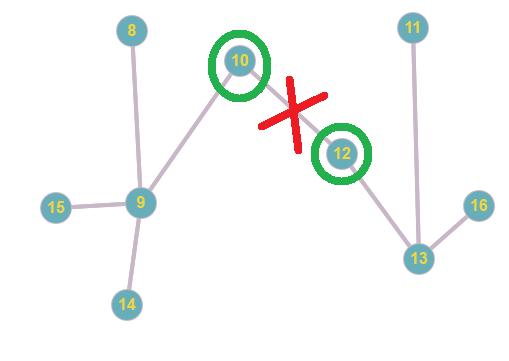
Se invece la lunghezza del percorso è pari *(figura 2)*, i possibili nodi medi sono due. A questo punto, eliminato l’arco che li collega, **vengono eseguite due visite generiche** a partire da ognuno di essi, che restituiscono il numero di elementi raggiungibili a partire dai due nodi; risulterà medio per il maggior numero di volte il nodo con il maggior valore di ritorno della visita. In caso di valore uguale, risulteranno entrambi medi.

Figura 1 – Rosso e blu sono i percorsi più lunghi

Nell’eventualità che il numero dei nodi medi per il maggior numero di volte all’interno di un sotto-grafo sia maggiore di uno, si utilizza come struttura d’appoggio una lista; al suo interno vengono memorizzati gli Id nodi medi all’interno del percorso ed il numero di volte che essi sono risultati tali.

Figura 2 - Tutti i percorsi hanno la stessa lunghezza massima pari a 6

Analizzati tutti i percorsi, avremo due liste: quella di appoggio contenente tutte le informazioni relative ai vari percorsi ed un’altra con all’interno gli Id dei nodi risultati medi. Per ogni elemento appartenente a quest’ultima lista, mi basterà tenere conto del numero di volte che è risultato medio all’interno dei vari percorsi. Sarà medio per il maggior numero di volte all’interno del grafo **il nodo (o i nodi) con questo valore 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 trovato il percorso più lungo ed il nodo con distanza maggiore dall’inizio del percorso, per ottenere la lista dei nodi appartenenti al percorso 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, se un nodo è già presente all’interno della lista dei nodi che risultano massimi, non verrà aggiunto nuovamente alla lista dei nodi da considerare.

**Strutture dati utilizzate**

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

**Analisi tempo teorico**

* leafDistance: | Eseguo una visita generica a partire dalla foglia, che viene considerata come radice dell’albero
* findLeaf: | Eseguo una visita generica a partire da un nodo qualsiasi del grafo e, nel caso peggiore, il numero delle foglie è uguale ad
* calculateSubNode: | 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: | Nel caso peggiore, la lunghezza del percorso è pari
* mediumNode: | Controllo tutti i nodi per eliminare quelli non connessi a nessun sotto-grafo, e nel caso peggiore il numero di foglie è

**Analisi tempo sperimentale**

L’analisi sperimentale conferma l’analisi teorica e mostra, inoltre, come la complessità dell’algoritmo sia strettamente legata al numero di foglie appartenenti al grafo. Il caso migliore si presenta con un grafo assimilabile ad un + 1 nodo disconnesso, in quanto è presente una singola foglia. Il caso peggiore, come è facile immaginare, si ha nel caso di un , dove 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**. I dati del grafico relativi ad , sono stati calcolati considerando il costo in secondi per il singolo nodo nel caso peggiore, risultato pari a **0.006528**.