**Descrizione algoritmo**

L’idea alla base dell’algoritmo è quella di considerare, in ogni sotto-grafo connesso, il percorso più lungo tra due foglie. Avendo a disposizione la lista dei nodi appartenenti al percorso, basta accedere agli elementi che stanno esattamente a metà percorso per ottenere il nodo che risulta medio per il maggior numero di coppie all’interno del sotto-grafo. Eseguendo quest’operazione per ogni sotto-grafo, e confrontando i vari valori ottenuti, l’algoritmo restituisce il nodo che risulta medio per il maggior numero di volte nell’intero grafo.

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, memorizzando la lista dei nodi appartenenti al percorso più lungo per ogni sotto-grafo in un array, una volta individuato il percorso con lunghezza maggiore, per ottenere l’Id del nodo dovrò solamente eseguire delle operazioni di slicing sull’array.

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)**
* **GraphAdjacentyLists**

**Analisi tempo teorico**

* leafDistance: | Eseguo una visita generica a partire dalla foglia, che viene considerata come radice dell’albero
* mediumNode: | Il numero delle foglie è nell’ordine di
* backToFather:
* controlloFunzione: in quanto devo considerare tutti i nodi

**Analisi tempo sperimentale**

L’analisi sperimentale conferma l’analisi teorica.