Scelte implementative

Per implementare il nostro il nostro progetto abbiamo fatto uso delle classi già definite GraphAdjacencyList, PQbinomialHeap e PQ_DHeap e dei loro rispettivi metodi.
Per la realizzazione della traccia abbiamo implementato due funzioni: il primo chiamato priorityVisit, mentre il secondo graphGenerator, che sfruttano altri tre metodi da noi progettati: getNodeMaxWeight, getId e getWeight.
Le code con priorità sono state cambiate per fare in modo che il massimo si trovasse sempre sulla radice.

getNodeMaxWeight

Questo metodo non prende nessun parametro e restituisce il nodo con il maggior peso. Viene inizializzato l'Id del nodo massimo a none, maxWeight a -1 (sfruttando la traccia che specifica di prendere i pesi dei nodi tutti strettamenti positi) e vengono messi dentro la variabile nodes tutti i nodi del nostro grafo non orientato. A questo punto tramite un ciclo for vengono controllati tutti i nodi e se il peso del nodo corrente è superiore a quello del massimo, maxWeight viene aggiornato. Alla fine del for troviamo il nodo con peso massimo.

GetId e GetWeight

GetId e getWeight sono due semplicissime funzioni implementate dentro graph che restituiscono rispettivamente l'Id e il peso di un nodo.

priorityVisit

priorityVisit è una funzione che prende come parametri l'oggetto grafo rappresentato mediante liste di adiacenza, la priorityQueue da usare per inserire i nodi visitati e il parametro d, inizializzato a 2, nel caso in cui si usasse una cosa con priorità di tipo DHeap. Se il parametro pq è uguale a True, viene utilizzato un heap binomiale, altrimenti, se viene preso un d > 1, viene usato un DHeap; se d risulta essere minore o uguale a 1 viene lanciata un'eccezione.
Il metodo inizia assegnando al parametro verticeMax il nodo con peso più alto e questo viene subito inserito all'interno della coda con priorità (come richiesto dalla consegna); a questo punto viene creata una lista di nodi già visitati, chiamata markedNodes al cui interno viene inserito il verticeMax (poiché è già stato visitato) e una lista (chiamata list) inizialmente vuota, che verrà ritornata alla fine del metodo per stampare il risultato della visita.
Adesso viene eseguito un while finché la coda non risulta vuota, viene inizializzato idNode con la radice della PQ e questo elemento viene inserito alla fine di list.
Dentro il while vengono presi tramite un for tutti i nodi adiacenti di IdNode e inseriti dentro AdjacentList; tramite un for vengono controllati tutti e, se tali nodi non sono presenti in markedNodes vengono inseriti nella coda con priorità e aggiunti alla lista dei nodi marcati.
finito il while viene ritornata la lista.

graphGenerator

graphGenerator permette di creare grafi randomici non orientati e connessi passandogli come parametri il numero di nodi e di archi che si vogliono inserire.
Naturalmente se si prova a inserire un numero di nodi minore o uguale a 1 oppure un numero di archi non appropriato, verrà lanciata un'eccezione.
Tramite un for vengono creati il numero di nodi inseriti dall'utente e gli vengono assegnati dei pesi randomici da 0 a 100.
A questo punto vengono collegati tutti i nodi tra di loro in modo casuale con il numero minimo di archi, per avere la certezza che il grafo sia connesso. Ora per inserire gli archi rimanenti (k = numero di archi richiesti – numero di archi inseriti), prendiamo due nodi casuali node1 e node2 che non sono già collegati tra loro e aggiungiamo un arco non orientato tra questi usando il metodo insertEdge.
Ritorno infine il grafo così generato.

Spiegazione della scelta di m1, m2, m3 e m4

Poiché al variare del numeri di nodi che un grafo può contenere, possono variare il numero minimo e il numero massimo di archi, abbiamo deciso di prendere m2 e m3 come due misure intermedie.
Per calcolare m2 e m3 abbiamo fatto il seguente ragionamento:
supponendo che n sia il numero di nodi abbiamo che:

m1 è il numero minimo di archi che il grafo può avere

m1 = n – 1

m4 è il numero massimo di archi che un grafo può avere: dalla teoria sappia che questo è pari a

$$m4 = \frac{\mathrm{n(n-1)(n-2)!}}{\mathrm{2(n-2)!}} \quad \rightarrow \quad m4 = \frac{\mathrm{n(n-1)}}{2}$$

ora facendo la differenza tra m4 e m1 possiamo calcolare la loro distanza e dividendola per 4 otteniamo l'offset:

$$d = \frac{\mathrm{m4-m1}}{4} = \frac{n^2-3n+2}{8}$$

prendendo ora questo valore e sommandolo a m1 otteniamo m2:

$$m2 = m1 + d = \frac{n^2 + 5n - 6}{8}$$

per m3 basta sottrarre a m4 il valore d:

$$\mathrm{m3} = \mathrm{m4} - \mathrm{d} = \frac{3n^2 - n - 2}{2a8}$$