Progetto a cura di Davide Montagno Bozzone

Graph<E>

Graph<E>, realizzato come grafo **orientato**, è stato ideato per essere utilizzato come rete sociale delle amicizie; attraverso i metodi implementati all'interno del progetto, si può facilmente interagire con i contenuti messi a disposizione;

Per una scelta personale del progetto, ciascuno grafo è gestito con un **HashMap**. Ho scelto questa implementazione poiché l'HashMap non è altro che una collezione di oggetti il cui scopo principale è quello di rendere veloci ed efficienti operazioni quali inserimento e ricerca di elementi. Un'altra scelta di progetto è stata quella di utilizzare eccezioni di tipo checked opportunamente definite all'interno del progetto; troviamo:

- AlreadyPresentException: Se esiste già un vertice o un arco
- NotPresentException: Se operiamo su vertici inesistenti, o archi inesistenti

Di seguito viene fornita una guida all'uso del progetto Graph<E>:

1. **MyGraph<E>** è la classe principale per la gestione del grafo. Implementa l'interfaccia **Graph<E>**; al suo interno troviamo i metodi principali per interagire col grafo e le sue relative istanze.
   1.1. Istanze:
      1.1.1. list (HashMap<Vertice(E), MyNode(E)>), utilizzata per gestire i vertici.
   1.2. Metodi:
      1.2.1. Costruttore: Inizializza this a vuoto
      1.2.2. aggiungiVertice: Aggiunge un vertice
      1.2.3. aggiungiArco: Aggiunge un arco
      1.2.4. rimuoviVertice: Rimuove un vertice
      1.2.5. rimuoviArco: Rimuove un arco
      1.2.6. numberOfVertici: Restituisce il numero di vertici presenti
      1.2.7. isInVertice: Controlla se è presente un vertice
      1.2.8. numberArchiOf: Restituisce il numero di archi uscenti di un dato vertice
      1.2.9. camminoMinimo: Restituisce il valore del cammino minimo tra due vertici (Se possibile)
      1.2.10. BFS: crea il cammino minimo tra due vertici
      1.2.11. inizialize: riporta i valori di default nei nodi;
      1.2.12. diametro: restituisce il valore massimo tra i cammini minimi
      1.2.13. getArchiUscenti: restituisce tutti gli amici di un dato vertice;
      1.2.14. getCommonArchiUscenti: restituisce tutti gli amici in comune tra i vertici passati come parametri

MyGraph<E> ha due strutture di appoggio che facilitano la creazione del grafo:

1. **MyNode<E>:** Questa classe è stata implementata per gestire gli archi di un dato vertice; implementa l'interfaccia **Node<E>**, al suo interno troviamo **1.3. Istanze:**
      1.3.1. Vertice
      1.3.2. Archi: Gestita come una lista di MyEdge<E>
      1.3.3. Padre: Di tipo Node<E>
      1.3.4. Colore: Utilizzato per realizzare la BFS
   1.4. Metodi:

1.4.1. getVertice: Restituisce il vertice corrente
1.4.2. aggiungiArco: Aggiunge un arco tra due nodi (Se possibile)
1.4.3. rimuoviArco: Rimuove un arco tra due nodi (Se possibile)
1.4.4. IsThereEdge: Controlla se è presente un arco tra due nodi
1.4.5. listOfArchi: restituisce la lista degli archi di un nodo (Se possibile)
1.4.6. GetPadre: Restituisce il padre del nodo corrente
1.4.7. IsVisited: Controlla se è stato visitato il nodo corrente
1.4.8. SetVisited: setta il colore
1.4.9. SetPadre: setta il padre
1.4.10. NumberOfArchi: restituisce il numero di archi uscenti presenti nel nodo.

2. **MyEdge<E>:** Questa classe è stata implementata per la creazione di un arco tra due nodi; implementa l'interfaccia **Edge<E>**; al suo interno troviamo:
   2.1. **Istanze:**
      2.1.1. Sorgente: di tipo Node<E> 2.1.2. Destinazione: di tipo Node<E>
   2.2. **Metodi:**
      2.2.1. Sorgente: restituisce il nodo sorgente
      2.2.2. Destinazione: Sostituisce il nodo destinazione
      2.2.3. esisteArco: Controlla se i valori passati coincidono con l'arco che stiamo controllando

Scelte di implementazione del progetto

1. **Scelta della gestione di 3 classi:**
   Ho scelto di implementare 3 classi, poiché era più facile gestire i 2 aspetti principali del progetto (Gestione vertice, Gestione archi del vertice).
2. **Scelta della gestione delle eccezioni:**
   Ho scelto di creare 2 eccezioni non fornite da Java, per segnalare all'utente errori specifici durante la manipolazione della struttura del grafo.
3. **Operazioni su valori null:**
   Non è possibile operare in alcun modo su oggetti "null".
4. **MyPerson**
   MyPerson non è stata descritta in precedenza poiché è una classe di appoggio per il test della rete sociale. Dunque la si descrive in poche parole qui: Implementa l'interfaccia Person; oltre al metodo costruttore, ha i metodi per cambiare il nome e il cognome (setNome, setCognome) ad una persona e si possono ottenere le sue informazioni tramite il metodo getInformation().
5. **Main**
   La classe Run_Social è la classe principale contenente il main. All'interno si provano tutte le funzioni di aggiunta e rimozione di un vertice, aggiunta e rimozione di un arco, vedere gli archi uscenti in comune tra due vertici ecc..

Davide Montagno Bozzone