A.A. 2016 - 17

Metodologie di Programmazione (canale A-L)
Homework 3 - consegna 26 aprile 2017

Definizioni. Un *grafo orientato* $\mathcal{G}(V, E)$ è costutuito da un insieme V di *vertici*, o *nodi*, ed un insieme E di *archi*. Ogni arco e ha associati un nodo $p(e) \in V$, il suo nodo di partenza, ed uno, $a(e) \in V$, di arrivo. Un *cammino orientato* è una sequenza $< e_1, \ldots, e_n >$ di archi, dove $a(e_i) = p(e_{i+1})$ per $i = 1, \ldots, n-1$. Se la sequenza è non vuota, i nodi $p(e_1)$ e $a(e_n)$ sono chiamati rispettivamente *inizio* e *fine* del cammino, e n è la sua lunghezza.

Una *rete* $\mathcal{N}(V, E, s, d)$ è un grafo orientato $\mathcal{G}_{\mathcal{N}}(V, E)$ con due nodi speciali $s \in V$ e $t \in V$, chiamati rispettivamente *source* (sorgente) e *target* (destinazione) della rete. Un *path*, è un cammino orientato (eventualmete ciclico) da souce a target.

Obiettivo. *Implementare in Java il tipo di dato astratto delle reti di T, dove T denota il tipo generico dei nodi della rete. Scrivere un programma che calcola il (un) path di lunghezza minima in una rete.*

Lavoro da svolgere. Scrivere una classe `MyNetwork <T>` che implementi la seguente interfaccia:

```
public interface Network <T> { // contratto delle reti

   public T source ();
      // restituisce il nodo sorgente di this;
      // null se la sorgente non e' settata

   public T target ();
      // restituisce il nodo destinazione di this;
      // null se la destinazione non e' settata

   public void setSource (T newsource) throws NoSuchNodeException;
      // setta la sorgente della rete a newsource;
      // lancia l'eccezione se newsource non e' un nodo della rete

   public void setTarget (T newtarget) throws NoSuchNodeException;
      // setta la destinazione della rete a newtarget;
```

```
      // lancia l'eccezione se newtarget non e' un nodo della rete

   public void addNode (T v);
      // aggiunge a this un nuovo nodo v.
      // Non fa nulla se v e' gia' un nodo della rete

   public void addEdge (T p, T a) throws NoSuchNodeException;
      // aggiunge a this un nuovo arco, con partenza p e arrivo a.
      // Lancia l'eccezione se p o a non appartengono alla rete.

   public List <T> shortestPath () throws NoSuchPathException;
      /*
      * restituisce una lista minimale <t1, ... tn> di oggetti di tipo T, tali che:
      * t1 e tn sono sorgente e destinazione della rete e, per ogni coppia (ti, t(i+1))
      * di elementi consecutivi nella lista, esiste in this (almeno) un arco con
      * partenza in ti e arrivo in t(i+1). Lancia l'eccezione se source o target
      * non sono settati o se non c'e' path che collega il primo al secondo
      */
}
```

Nota sull'uguaglianza. Alcuni metodi, ad esempio `addNode`, richiedono la verifica della presenza di un nodo nella rete. Questo richiede un confronto fra oggetti. Ma *quand'è che due oggetti sono uguali*? Senza andare troppo sul filosofico, adottiamo (*e verrà adottato nella correzione dell'homework!*) il seguente criterio: `a` e `b` sono uguali quando a.equals(b) è vero. Attenzione:

```
    public static void main(String[] args) {
        Integer a = new Integer(7);
        Integer b = new Integer(7);
        System.out.println(a == b); // stampa false
        System.out.println(a.equals(b)); // stampa true

    }
```

Questo avviene perché la classe `Integer` ridefinisce il metodo `equals` della classe `Object`, che è:

```
public boolean equals(Object obj) {
    return (this == obj);
}
```

Dunque, se volete fare il vostri test su una rete di `Gnat`, una classe di vostra fantasia, allora, se volete considerare uguali due gnat nel senso in cui lo sono `a` e `b` dell'esempio, allora dovete ridefinire in modo opportuno il metodo `equals` che `Gnat` eredita da `Object`.