```
public class Esame {

	public static final int MAX_ELEM = 1024;

	public static int contaTratte(Tratta[] tratte){
		int count = 0;
		while (count < MAX_ELEM && tratte[count] != null) {
			count++;
		}
		return count;
	}

	/*
	 * A.
	 * Il metodo deve calcolare la costo medio al chilometro,
espresso in EURO (€), di tutta la rete autostradale gestita.
	 */
	public static double costoRete(Tratta[] tratte) {
		double costo = 0.0;
		double distanza = 0.0;
		double media = 0.0;
		int n = contaTratte(tratte);
		int nt = 0;
		for (int i = 0; i < n; i++) {
			boolean conteggiato = false;
			for (int j = 0; j < i && !conteggiato; j++) {
				conteggiato =
tratte[i].getCodice().equals(tratte[j].getCodice());
			}
			if (!conteggiato) {
				nt++;
				costo +=tratte[i].costo
				distanza +=tratte[i].distanza;
			}
		}
		if (nt > 0) {
			media = costo/distanza;
		}
		return costo;
	}

	// scambia l'elemento in posizione i-esima con quello in
posizione j-esima
	private static void scambiaTratte(Tratta[] t, int i, int
j){
		Tratta tmp = t[i];
		t[i] = t[j];
		t[j] = tmp;
	}

	// scambia l'elemento in posizione i-esima con quello in
posizione j-esima
	private static void scambiaPassaggi(Passaggio[] p, int i,
int j){
		Passaggio tmp = p[i];
		p[i] = p[j];
```

```
            p[j] = tmp;
      }

      /*
       * B.
       * Il metodo deve ordinare, nell'array parallelo, gli
elementi in maniera crescente o decrescente
       * a seconda del valore dell'omonimo parametro, usando
come criterio la targa del veicolo che ha
       * percorso una tratta e poi il codice della tratta
stessa.
       * Le tratte senza passaggi associati vanno poste in cima
all'array in caso di ordinamento crescente
       * ed in fondo all'array in caso di ordinamento
decrescente.
       */
      public static void ordinaDB(Tratta[] tratte, Passaggio[]
passaggi, boolean crescente) {
            int n = contaTratte(tratte);
            for (int i = 0; i < n-1; i++) {
                  for (int j = i + 1; j < n; j++) {
                        if (
                              (
                                    crescente && (
                                          passaggi[i] == null || (
                                                passaggi[j] != null
&& (

      passaggi[j].targa.compareToIgnoreCase(passaggi[i].targa)
> 0 ||
                                                      (

      passaggi[j].targa.compareToIgnoreCase(passaggi[i].targa)
== 0 &&

      tratte[j].getCodice().compareToIgnoreCase(tratte[i].getCo
dice()) > 0
                                                      )
                                                )
                                          )
                                    )
                              ) || (
                                    !crescente && (
                                          passaggi[j] == null || (
                                                passaggi[i] != null
&& (

      passaggi[i].targa.compareToIgnoreCase(passaggi[j].targa)
> 0 ||
                                                      (

      passaggi[i].targa.compareToIgnoreCase(passaggi[j].targa)
== 0 &&

      tratte[i].getCodice().compareToIgnoreCase(tratte[j].getCo
dice()) > 0
                                                      )
```

```
                                                )
                                          )
                                    )
                              )
                        ){
                              scambiaTratte(tratte, i, j);
                              scambiaPassaggi(passaggi, i, j);
                        }
                  }
            }
      }

      /*
       * C.
       * Il metodo deve ritornare un array contenente tutte le
targhe dei veicoli che hanno effettuato
       * passaggi senza pagare il pedaggio.
       * Se un veicolo ha effettuato più passaggi senza pagare
il pedaggio deve comparire nell'array una sola volta.
       */
      public static String[] trasgressori(Tratta[] tratte,
Passaggio[] passaggi) {
            int n = contaTratte(tratte);
            int nt = 0, it = 0;
            String[] t = null;
            for (int i = 0; i < n; i++) {
                  if (passaggi[i] != null &&
passaggi[i].titoloViaggio == 'N') {
                        boolean conteggiato = false;
                        for (int j = 0; j < i && !conteggiato;
j++) {
                              conteggiato = passaggi[j] != null
&& passaggi[j].titoloViaggio == 'N' &&

      passaggi[i].targa.equals(passaggi[j].targa);
                        }
                        if (!conteggiato) {
                              nt++;
                        }
                  }
            }
            if (nt > 0) {
                  t = new String[nt];
                  for (int i = 0; i < n; i++) {
                        if (passaggi[i] != null &&
passaggi[i].titoloViaggio == 'N') {
                              boolean conteggiato = false;
                              for (int j = 0; j < i && !
conteggiato; j++) {
                                    conteggiato = passaggi[j] !=
null && passaggi[j].titoloViaggio == 'N' &&

      passaggi[i].targa.equals(passaggi[j].targa);
                              }
                              if (!conteggiato) {
                                    t[it] = passaggi[i].targa;
                                    it++;
```

```
                        }
                    }
                }
            }
            return t;
      }

      /*
       * D.
       * Il metodo deve inserire nel database specificato dai
parametri "tratte" e "passaggi"
       * un nuovo passaggio (individuato dal numero di targa,
titolo di viaggio, data di entrata e di uscita)
       * associato ad una tratta che non abbia un passaggio
associato oppure duplicando la tratta stessa
       * in caso contrario.
       * Il metodo deve restituire true o false a seconda del
fatto che sia stata trovata almeno una
       * tratta associata al codice passato come parametro nel
database. Se non esiste alcuna tratta
       * associata al codice passato come parametro,
l'inserimento non deve essere effettuato.
       * L'inserimento deve mantenere l'archivio ordinato come
nel punto precedente ed in uno stato
       * consistente.
       */
     public static boolean effettuaPassaggio(Tratta[] tratte,
Passaggio[] passaggi,
            String codice, String targa, char titoloViaggio,
String dataEntrata, String dataUscita) {
            boolean ret = false;
            int n = contaTratte(tratte);
            int sectionIdx = n;
            // Si cerca il codice della tratta.
            for (int i = 0; i < n && sectionIdx == n; i++) {
                  if (tratte[i].getCodice().equals(codice)) {
                        sectionIdx = i;
                  }
            }
            /* Inserimento possibile se: codice trovato e, c'è
spazio nell'array parallelo oppure
             * la tratta è senza passaggi associati.
             */
            ret = (sectionIdx != n) && (n < MAX_ELEM ||
passaggi[sectionIdx] == null);
            if (ret) {
                  Passaggio p = new Passaggio(targa,
titoloViaggio, dataEntrata, dataUscita);
                  if (passaggi[sectionIdx] == null) {   //
Tratta senza passaggio associato.
                        passaggi[sectionIdx] = p;
                  }
                  else {     // Inserimento in coda.
                        tratte[n] = tratte[sectionIdx];
                        passaggi[n] = p;
                  }
                  // Si ordina il nuovo DB.
```

```
                ordinaDB(tratte, passaggi, false);
            }
            return ret;
      }

      public static void stampaDB(Tratta[] tratte, Passaggio[]
passaggi){
            int n = contaTratte(tratte);
            for (int i=0; i < n; i++){
                  if (passaggi[i] != null){
                        System.out.println(tratte[i] + " => " +
passaggi[i]);
                  }
                  else {
                        System.out.println(tratte[i]);
                  }
            }
      }

      public static void main(String[] args) {

            Tratta[] sections = new Tratta[MAX_ELEM];
            Passaggio[] passes = new Passaggio[MAX_ELEM];

            sections[0] = new Tratta("PNVA12", "Pisa Nord",
"Viareggio", 11.7, 2.0, "A12");
            sections[1] = new Tratta("GEVA12", "Genova Est",
"Viareggio", 131.9, 15.0, "A12");
            sections[2] = new Tratta("FNBCA1", "Firenze Nord",
"Bologna Casalecchio", 85.4, 8.1, "A1");
            sections[3] = new Tratta("PNFNA11", "Pisa Nord",
"Firenze Nord", 76.4, 6.1, "A11");
            sections[4] = sections[3];
            sections[5] = sections[0];
            sections[6] = sections[2];
            sections[7] = sections[2];

            passes[5] = new Passaggio("AJ632RR", 'T',
"28/04/2018 07:02", "28/04/2018 07:15");
            passes[0] = new Passaggio("PG345ZQ", 'T',
"01/05/2018 20:10", "01/05/2018 20:15");
            passes[6] = new Passaggio("AS768ED", 'B',
"18/05/2018 08:36", "18/05/2018 09:45");
            passes[7] = new Passaggio("HJ234FR", 'N',
"01/06/2018 09:05", "01/06/2018 08:22");
            passes[3] = new Passaggio("HJ234FR", 'N',
"01/06/2018 09:36", "01/06/2018 09:05");
            passes[4] = new Passaggio("QR987LK", 'T',
"10/06/2018 22:45", "11/06/2018 00:05");
            passes[2] = new Passaggio("QR987LK", 'T',
"11/06/2018 00:05", "11/06/2018 01:23");

            System.out.println("\nA.");
            System.out.println("Il costo medio al chilometro di
tutta la rete autostradale gestita è di €: " +
                        costoRete(sections));
```

```
            System.out.println("\nB.");
            System.out.println("Database PRIMA
dell'ordinamento:");
            stampaDB(sections, passes);
            ordinaDB(sections, passes, true);
            System.out.println("\nDatabase ordinato in maniera
crescente:");
            stampaDB(sections, passes);
            ordinaDB(sections, passes, false);
            System.out.println("\nDatabase ordinato in maniera
decrescente:");
            stampaDB(sections, passes);

            System.out.println("\nC.");
            String[] t = trasgressori(sections, passes);
            if (t.length > 0) {
                  System.out.println("Lista Tragressori:");
                  for (int i = 0; i < t.length; i++)
                  {
                        System.out.println(t[i]);
                  }
            }
            else {
                  System.out.println("Nessun Trasgressore!");
            }

            System.out.println("\nD.");
            if (effettuaPassaggio(sections, passes, "PNFNA11",
                        "HJ234FR", 'B', "14/01/2019 07:11",
"14/01/2019 07:59")) {
                  System.out.println("Operazione effettuata,
nuovo database:");
                  stampaDB(sections, passes);
            }
            else {
                  System.out.println("Operazione non
effettuata!");
            }

      }

}
```