Kurier-Express-Dienst

Anbieter von Kurier-Express-Diensten transportieren vornehmlich Pakete mit geringem Gewicht und Volumen. Depots dienen als Sammel- und Verteilzentren für die Pakete.
Die Pakete von verschiedenen Kunden gehen im Depot ein, wo sie sortiert und nach Empfangsdepots gebündelt werden. Anschließend werden die Pakete zu den jeweiligen Empfangsdepots transportiert, wo sie dann am folgenden Tag zu Auslieferungstouren zusammengestellt und schließlich mittels Kleintransporter zum Kunden ausgeliefert werden.

Aufgaben

1 In Material 1 ist das UML-Klassendiagramm für einen solchen Dienst dargestellt.

1.1 Beschreiben Sie die Klasse *Route* des UML-Diagramms.

(4 BE)

1.2 Zeichnen Sie die Beziehungen zwischen den Klassen in das Diagramm ein und erläutern Sie diese.

(4 BE)

2 Die Pakete werden vor jeder Auslieferungstour gescannt und in die Packliste eingefügt. Bei der Zustellung werden die Pakete erneut gescannt und aus der Packliste entfernt.
Implementieren Sie die Methoden *entferneErstesPaket()* und *entfernePaket(…)* der Klasse *Packliste*.

(8 BE)

3 Der Quellcode der Methoden *berechneRouteUndLänge()* und *berechneWerte(…)* der Klasse *Route* befindet sich in Material 2 bzw. Material 3.

3.1 Analysieren Sie die Methode *berechneRouteUndLänge()* der Klasse *Route*.

(4 BE)

3.2 Analysieren Sie die Zeilen 13–21 der Methode *berechneWerte(…)* der Klasse *Route*.

(6 BE)

4 In der Klasse *Route* sollen die Adressen der Empfänger nach der Postleitzahl sortiert werden.
Implementieren Sie die Methode *sortiereNachPLZ()* der Klasse *Route*.

Hinweis:

```
public int compareTo(String anotherString)
```

String1.compareTo(String2) vergleicht *String1* mit *String2*. Das Ergebnis ist kleiner als 0, falls *String1* lexikographisch vor *String2* kommt, und größer als 0, falls *String1* lexikographisch nach *String2* kommt. Falls die beiden Strings gleich sind, ist das Ergebnis 0.

(4 BE)

5 Empfangene Pakete werden in den Depots mit dem Algorithmus *Distributionsort* sortiert. Die Pakete werden anhand der Postleitzahl des Empfängers auf zehn Paletten mit den Ziffern 0 bis 9 verteilt. Man verteilt zuerst anhand der letzten Ziffer, sammelt die Pakete unter Beibehaltung der Ordnung auf den Paletten wieder ein, verteilt die Pakete anschließend nach der vorletzten Ziffer neu usw.

5.1 In Material 4 werden die ersten drei Schritte des Algorithmus für zehn Beispieladressen dargestellt. Bestimmen Sie jeweils die Reihenfolge der Adressen nach den letzten beiden Sortiervorgängen.
Hinweis: Die Angabe der Postleitzahlen reicht aus.

(2 BE)

5.2 Vergleichen Sie die beiden Sortierverfahren aus Aufgabe 4 und Aufgabe 5.1 hinsichtlich der asymptotischen Zeitkomplexität.

(3 BE)

Material 1

UML-Klassendiagramm zur Auslieferung

Hinweise zur Klasse Route

- Im Feld *dieAdressen* sind an Index 0 die Adresse des Depots und an Index n die Empfangsadresse des Feldes *dieEmpfänger* an Index n-1 gespeichert.
- Im zweidimensionalen Feld *Entfernung* werden die jeweiligen ermittelten Entfernungen der Empfangsadressen zueinander gespeichert. Der Wert Entfernung[i][j] gibt die Entfernung der Adresse i zur Adresse j an.

Material 2

Methode *berechneRouteUndLänge()* der Klasse *Route*

```
01 private void berechneRouteUndLänge(){
02   int Anzahl = dieAdressen.length;
03   int Depot = 0;
04   boolean[] besucht = new boolean[Anzahl];
05   besucht[Depot] = true;
06   int[] eineRoute = new int[Anzahl];
07   Länge = Double.MAX_VALUE;
08   int Index = 1;
09   berechneWerte(Depot, 1, 0.0, besucht, Index, eineRoute);
10 }
```

Hinweis zur Klasse Double
Die Konstante MAX_VALUE enthält den größten positiven endlichen Wert des Datentyps double.

Material 3

Methode *berechneWerte(…)* der Klasse *Route*

```
01 private void berechneWerte(int Ort, int Anzahl, double LängeNeu,
02                            boolean[] besucht, int Index,
03                            int[] eineRoute){
04   if (Anzahl == dieAdressen.length){               // Alle Orte besucht,
05     LängeNeu = LängeNeu + Entfernung[Ort][0];      // zurück zu Depot.
06     if (LängeNeu < Länge) {                        // Ist neue Länge
07       for (int i = 0; i < eineRoute.length; i++) { // kürzer, wird die
08         optimaleRoute[i] = eineRoute[i];           // aktuelle Route und
09       }                                            // deren Länge in den
10       Länge = LängeNeu;                            // Attributen
11     }                                              // gespeichert
12   }
13   for (int i = 1; i < dieAdressen.length; i++){
14     if (besucht[i] == false){
15       besucht[i] = true;
16       eineRoute[Index] = i;
17       berechneWerte(i, Anzahl + 1, LängeNeu + Entfernung[Ort][i],
18                     besucht, Index + 1, eineRoute);
19       besucht[i] = false;
20     }
21   }
22 }
```

Material 4

Beispieladressen zur Veranschaulichung des Distributionsort-Algorithmus

35037 Marburg
35390 Gießen
64283 Darmstadt
63065 Offenbach
61250 Usingen
61169 Friedberg
34117 Kassel
36043 Fulda
63679 Schotten
65185 Wiesbaden

Adressen vor dem Sortieren

35390 Gießen
61250 Usingen
64283 Darmstadt
36043 Fulda
63065 Offenbach
65185 Wiesbaden
35037 Marburg
34117 Kassel
61169 Friedberg
63679 Schotten

Adressen sortiert nach der letzten Ziffer

34117 Kassel
35037 Marburg
36043 Fulda
61250 Usingen
63065 Offenbach
61169 Friedberg
63679 Schotten
64283 Darmstadt
65185 Wiesbaden
35390 Gießen

Adressen sortiert nach der vorletzten Ziffer

35037 Marburg
36043 Fulda
63065 Offenbach
34117 Kassel
61169 Friedberg
65185 Wiesbaden
61250 Usingen
64283 Darmstadt
35390 Gießen
63679 Schotten

Adressen sortiert nach der dritten Ziffer