Aufgabe 2: Vollgeladen

Team-ID: 00070

Team: Gaußgamz

Bearbeiter/-innen dieser Aufgabe:
Manuel Pielka, Benedict Weis, Tina Balaban

30. September 2021

Inhaltsverzeichnis

Lösungsidee

Das Programm soll für eine vorgegebene Strecke eine Route, bestehend aus 4 Hotelstopps, die eine möglichst hohe Bewertung haben, bestimmen. Dafür bestimmt es zunächst eine provisorische Route, mit der man das Ziel erreichen kann, und tauscht, sofern nötig, die jeweiligen Hotels mit einem anderen Hotel mit besserer Bewertung aus.

Umsetzung

Da die Reise nicht länger als 5 Tage dauern soll und man pro Tag nicht länger als 6 Stunden fahren soll, kann pro Tag nur eine Distanz von max. 360 (60*6) Autominuten zurückgelegt werden.

Zunächst wird mit der gegebenen Liste der Hotels eine provisorische Route erstellt. Diese besteht aus 4 aus der Liste ausgewählten Hotels, die alle möglichst weit innerhalb der 360 Autominuten-Distanz voneinander entfernt sind. Somit existiert bereits eine Route, mit der man das Ziel erreichen würde.

Danach werden die ausgewählten Hotels, beginnend mit dem letzten Stopp der Route, mit den umliegenden Hotels in der Reichweite verglichen und ggf. mit einem besser bewerteten Hotel ausgetauscht.

Hierbei muss beachtet werden, dass das vorherige Hotel durch den Tausch nicht außerhalb der Reichweite gelangen würde, sollte dieses durch einen Tausch nicht wieder in die Reichweite rücken können.

Dieser Optimierungsprozess wird solange wiederholt bis man kein Hotel mehr tauschen kann. Folglich sollte die finale Version der Route die Optimale sein und wird ausgegeben.

Beispiele

Nehmen wir die Daten von „hotels1.txt“ als Beispiel:

Das Programm erstellt eine provisorische Route mit folgenden Hotels:

359 2.6

687 4.4

1008 2.6

1360 2.8

Nun wird überprüft, ob es innerhalb der Reichweite, beginnend beim Ende, ein besser bewertetes Hotel gibt, als das bisher ausgewählte Hotel.

Dieses Hotel gibt es zwar, allerdings kann es nicht an den Rest der Route angefügt werden, weil es zu weit weg ist. Daher wird es erst einmal nicht beachtet. Dann wird in der Reichweite des letzten Hotels ein eventuell besserer Kandidat für das vorletzte Hotel gesucht. Das Hotel 1007 2.8 liegt sowohl in der Reichweite des aktuellen Vorgängers, als auch in der des Nachfolgers und hat eine bessere Bewertung als 1008 2.6. Somit kann es getauscht werden.

Danach werden die restlichen beiden Hotelstopps nach dem gleichen Muster verglichen und ggf. ausgetauscht. Dann wird die Optimierung wieder am Ende der Route angefangen. Diese Schleife läuft solange durch, bis in einem Durchlauf nichts mehr geändert wurde.

Die finale und somit ausgegebene Route lautet:

Selected Hotels:

Hotel 0: 347 2.7

Hotel 1: 687 4.4

Hotel 2: 1007 2.8

Hotel 3: 1360 2.8

Average Rating: 3.175

Die weiteren Beispieldateien werden mit der selben Vorgehensweise bearbeitet, daher hier nur ihre Ausgaben:

hotels2.txt:

Selected Hotels:

Hotel 0: 341 2.3

Hotel 1: 700 3.0

Hotel 2: 1053 4.8

Hotel 3: 1380 5.0

Average Rating: 3.775

hotels3.txt:

Selected Hotels:

Hotel 0: 359 4.6

Hotel 1: 717 0.3

Hotel 2: 1076 3.8

Hotel 3: 1433 1.7

Average Rating: 2.6

hotels4.txt:

Selected Hotels:

Hotel 0: 340 4.6

Hotel 1: 676 4.6

Hotel 2: 1032 4.9

Hotel 3: 1301 5.0

Average Rating: 4.775

hotels5.txt:

Selected Hotels:

Hotel 0: 280 5.0

Hotel 1: 636 5.0

Hotel 2: 987 5.0

Hotel 3: 1271 5.0

Average Rating: 5.0

Die Daten der von uns erstellten Datei hotels6.txt sind eine Ausnahme: hierbei tritt der Fall ein, dass sich unter den gegebenen Bedingungen keine Route erstellen lässt. In dieser Beispieldatei ist nur das erste Hotel erreichbar, alle anderen liegen außerhalb der Reichweite.

Daher gibt das Programm aus:

„Error: No possible route found“

Das Programm kann aus jeder beliebigen Datei, solange das korrekte Format und die korrekten Datentypen vorliegen und sich aus den Daten eine Route konstruieren lässt, eine Route erstellen.

Sollte eine dieser Bedingungen nicht erfüllt sein, wird eine entsprechende Error-Meldung ausgegeben.

Quellcode

```
public static void main(String[] args) {
         […]
         int num_of_hotels;
         int totalTime;
          ArrayList<Hotel> hotels = new ArrayList<Hotel>();
         […]
         if (totalTime > 1800) {
         System.out.println("Error: Total route time exceeds theoretical maximum that can be completed");
      return;
    }
    float averageRating;
    ArrayList<Hotel> selectedHotels = createRoute(totalTime, hotels);

    if (selectedHotels == null)
      return;

    selectedHotels = optimizeRoute(totalTime, selectedHotels, hotels);
}
static ArrayList<Hotel> createRoute(int totalTime, ArrayList<Hotel> hotels) {
         int currentTravelTime = 0;
         ArrayList<Hotel> selectedHotels = new ArrayList<Hotel>();
         while (currentTravelTime < totalTime - 360) {
                  Hotel currentHotel = getFarthestHotelWithinRange(hotels, currentTravelTime);
                  selectedHotels.add(currentHotel);
                  currentTravelTime = currentHotel.distance;
                  if (selectedHotels.size() > 4) {
                           System.out.println("Error: No possible route found");
                           return null;
                  }
```

```
        }
        return selectedHotels;
}
static ArrayList<Hotel> optimizeRoute(int totalTime, ArrayList<Hotel> selectedHotels, ArrayList<Hotel> hotels) {
        int previousHotelDistance = 0;
        int nextHotelDistance = 0;
        float averageRating;
        ArrayList<Hotel> previousRoute = new ArrayList<Hotel>();
        while (!previousRoute.equals(selectedHotels)) {
        previousRoute = (ArrayList<Hotel>) selectedHotels.clone();
        for (int i = selectedHotels.size() - 1; i > -1; i--) {
        if (i == 0) {
                previousHotelDistance = 0;
        } else {
                previousHotelDistance = selectedHotels.get(i - 1).distance;
        }
        if (i == selectedHotels.size() - 1) {
        nextHotelDistance = totalTime;
        } else {
        nextHotelDistance = selectedHotels.get(i + 1).distance;
        }
         Hotel currentHotel = selectedHotels.get(i);
        for (Hotel h : getRemainingHotels(hotels, selectedHotels)) {
                if (h.distance - previousHotelDistance < 360 && h.distance - previousHotelDistance > 0
                && nextHotelDistance - h.distance < 360 && nextHotelDistance - h.distance > 0
                && currentHotel.rating < h.rating) {
                        currentHotel = h;
                }
        }
        selectedHotels.set(i, currentHotel);
        averageRating = calculateAverageRating(selectedHotels);
        while (selectedHotels.size() < 4) {
                averageRating = calculateAverageRating(selectedHotels);
                Hotel bestRemaining = getBestRemainingHotel(hotels, selectedHotels);
                if (bestRemaining.rating > averageRating)
                        selectedHotels.add(bestRemaining);
                else
                        break;
        }
        Collections.sort(selectedHotels);
        return selectedHotels;
}
static Hotel getFarthestHotelWithinRange(ArrayList<Hotel> list, int currentTravelTime) {
    Hotel current = null;
    for (Hotel h : list) {
      if (current == null)
        current = h;
```

```
      if (current.distance <= h.distance && h.distance - currentTravelTime <= 360
          && h.distance - currentTravelTime > 0)
        current = h;
    }
    return current;
}

static ArrayList<Hotel> getRemainingHotels(ArrayList<Hotel> allHotels, ArrayList<Hotel> selectedHotels) {
    ArrayList<Hotel> remainingHotels = allHotels;
    for (Hotel h : selectedHotels) {
      remainingHotels.remove(h);
    }
    return remainingHotels;
}
[…]
static Hotel getBestRemainingHotel(ArrayList<Hotel> allHotels, ArrayList<Hotel> selectedHotels) {
    ArrayList<Hotel> remainingHotels = getRemainingHotels(allHotels, selectedHotels);
    Hotel bestHotel = null;
    for (Hotel h : remainingHotels) {
      if (bestHotel == null || bestHotel.rating < h.rating)
        bestHotel = h;
    }
    return bestHotel;
}
```