Aufgabe 2:
Vollgeladen

Team-ID: 00370

Team: Linus Schumann

Bearbeiter/-innen dieser Aufgabe:

Linus Schumann

15.11.2021

Inhaltsverzeichnis

1_Lösungsidee 2
1.1_Funktionsweise von Bubble Sort 2
2_Umsetzung 2
2.1_Allgemeines 2
2.2_calculateNextHotel-Methode 2
2.2.1_Ende der Reise 3
2.3_calculateMinimalRating-Methode 3
2.4_Bubble Sort 3
3_Beispiele 3
3.1_Hotels 1 3
3.2_Hotels 2 4
3.3_Hotels 3 4
3.4_Hotels 4 4
3.5_Hotels 5 5
4_Quellcode 5

1_Lösungsidee

Um das Problem zu lösen und die besten Hotels zu finden, werden mögliche Wege ausprobiert und der Weg mit der größten niedrigsten Bewertung gespeicher. Um nicht jeden Weg ausprobieren zu müssen, wird ein einzelner Versuch abgebrochen, wenn das Ziel nicht mehr erreicht werden kann oder das nächste Hotel zu weit entfernt ist. Dabei muss beachtet werden, dass für eine 5-tägige Reise nur 4 Hotels benötigt werden.

Auch wird das Programm insofern erweitern, dass nicht nur das Hotel mit niedrigster Bewertung, sondern auch das Hotel mit der zweit, dritt und viert niedrigsten Bewertung betrachtet wird. Dazu soll die Liste der ausgewählten Hotels mit Hilfe des Algorithmus „Bubble Sort“ sortiert werden.

1.1_Funktionsweise von Bubble Sort

Der Bubble Sort Algorithmus hat eine Laufzeit von $O(n^2)$. Dies ergibt sich dadurch, dass für jedes Element so viele Schritte notwendig sind, wie es Elemente gibt. Allgemein funktioniert der Algorithmus so, dass er für jedes Element einmal durch die Liste iteriert. Dabei wird immer überprüft, ob das aktuelle Element größer ist als das nächste Element. In diesem Fall würde das Element mit Hilfe eines Dreieckstauschs getauscht. Jedes Mal, wenn die Liste durchlaufen wurde, hat man somit das größte Element am Ende. Dies wird gespeichert und das Ende der Liste wird eins nach vorne gesetzt.

2_Umsetzung

2.1_Allgemeines

Das Programm wurde in Java geschrieben und mit der Java-JDK 17 kompiliert. Im Ordner „executables“ lässt sich die kompilierte .jar-Datei finden und mit der Batchdatei (Windows) oder dem Shell-Script (Linux und MacOS) ausführen. Außerdem lassen sich im Ordner „beispieldaten“ alle in dieser Dokumentation aufgeführten Beispiele wiedergefunden werden.

Im Ordner „source“ lassen sich folgende Java Klassen wiederfinden:

- Main: Die Klasse „Main“ ist die Hauptklasse sie übernimmt die Dateiauswahl und gibt den Dateinamen an die andere Klasse weiter
- a2: Die Klasse „a2“ (Aufgabe 2) übernimmt die Verarbeitung der Daten und die Ausgabe der Ergebnisse

Die Umsetzung der Lösungsidee, wurde wie oben erläutert, in der Klasse „a2“ umgesetzt und beginnt mit dem Einlesen der einzelnen Hotels. Dabei werden die Entfernung und die Bewertung in zwei unterschiedlichen Listen gespeichert. Danach wird die Funktion „calculateNextHotel“ aufgerufen und danach werden die besten Hotels ausgegeben. Dabei wird zum einem die Position der einzelnen Hotels sowie die Bewertungen der einzelnen Hotels ausgegeben. Zuletzt wird noch die niedrigste Bewertung ausgeben.

2.2_calculateNextHotel-Methode

Die Berechnung der Hotels wird durch eine rekursive Funktion übernommen, an die die Indexe der bereits in diesem Versuch verwendeten Hotels, der aktuelle Tag und die vergangene Zeit übergeben wird. Zuerst wird in dieser Funktion überprüft ob schon 5 Tage vergangen sind. Der weitere Vorgang, wenn dies nicht der Fall ist, wird in „2.2.1_Ende der

Reise“ beschrieben. Wenn allerdings noch nicht 5 Tage vergangen sind, dann wird der nächste Tag mit allen Hotels berechnet, die nicht weiter als 360 km entfernt sind, das Ziel noch erreicht werden kann und die Hotel Bewertung nicht niedriger ist als die aktuell beste niedrigste Bewertung.

2.2.1_Ende der Reise

Wie in 2.2 erklärt wird dieser Teil nach 5 Tagen aufgerufen und es wird berechnet, ob dieser Weg besser wäre. Da schon vorher überprüft wurde, ob der Weg möglich ist, muss dies nicht nochmal geprüft werden. Zuerst wird die niedrigste Bewertung mit der besten niedrigsten Bewertung verglichen. Dabei wird die niedrigste Bewertung mit der Funktion „calculateMinimalRating“ berechnet. Diese Funktion wird genauer in 2.3 beschrieben. Wenn diese Bewertung besser ist, dann wird die niedrigste Bewertung und die dazugehörigen Hotels abgespeichert. Wenn diese Bewertung gleich ist, dann wird die 2. Beste Bewertung ebenfalls mit der Funktion „calculateMinimalRating“ berechnet, verglichen und dementsprechend abgespeichert, falls es besser war. Falls die Bewertung immer noch gleich sein sollte, wird dieser Schritt für das 3. Und 4. Hotel ebenfalls wiederholt.

2.3_calculateMinimalRating-Methode

Diese Funktion sortiert die Liste und gibt den Wert an einem Index der Liste zurück. Als Parameter wird eine Liste und der Index übergeben. Genutzt wird diese Funktion dazu das erste, zweite, dritte… Minimum zurückzugeben. Um die Liste zu sortieren, wird die Funktion „bubbleSort“ (2.4) genutzt, die den gleichnamigen Algorithmus nutzt.

2.4_Bubble Sort

Diese Funktion sortiert die Liste mit dem Bubble-Sort Algorithmus, der in 1.1 erklärt wurde.

3_Beispiele

3.1_Hotels 1

Ausgabe „hotels1.txt“

```
These hotels are the bests:
2. Hotel at: 347 min  Rating: 2.7
6. Hotel at: 687 min  Rating: 4.4
7. Hotel at: 1007 min  Rating: 2.8
10. Hotel at: 1360 min  Rating: 2.8
Lowest-Rating: 2.7
```

3.2_Hotels 2

Ausgabe „hotels2.txt“

```
These hotels are the bests:
2. Hotel at: 341 min  Rating: 2.3
9. Hotel at: 700 min  Rating: 3.0
14. Hotel at: 1053 min  Rating: 4.8
24. Hotel at: 1380 min  Rating: 5.0
Lowest-Rating: 2.3
```

3.3_Hotels 3

Ausgabe „hotels3.txt“

```
97. Hotel at: 359 min  Rating: 4.6
195. Hotel at: 717 min  Rating: 0.3
297. Hotel at: 1076 min  Rating: 3.8
400. Hotel at: 1433 min  Rating: 1.7
Lowest-Rating: 0.3
```

3.4_Hotels 4

Ausgabe „hotels4.txt“

```
These hotels are the bests:
96. Hotel at: 340 min  Rating: 4.6
211. Hotel at: 676 min  Rating: 4.6
331. Hotel at: 1032 min  Rating: 4.9
425. Hotel at: 1301 min  Rating: 5.0
Lowest-Rating: 4.6
```

3.5_Hotels 5

Ausgabe „hotels5.txt“

```
These hotels are the bests:
284. Hotel at: 317 min  Rating: 5.0
580. Hotel at: 636 min  Rating: 5.0
912. Hotel at: 987 min  Rating: 5.0
1177. Hotel at: 1286 min  Rating: 5.0
Lowest-Rating: 5.0
```

4_Quellcode

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class a2 {
    //___Create Variables___//
    static int[] hotelDistance;
    static double[] hotelRating;
    static int totalTravelTime;
    static int numberOfHotels;
    static int dailyTravelTime = 360;
    static int[] bestHotels;
    static double bestHotelRating = 0;
    public static void calculateHotels(String filename) throws
FileNotFoundException {
        //___Read-file___//
        File file = new File(filename);
        Scanner scanner = new Scanner(file);
        numberOfHotels = scanner.nextInt();
        totalTravelTime = scanner.nextInt();
        hotelDistance = new int[numberOfHotels];
        hotelRating = new double[numberOfHotels];
        for(int i = 0; i < numberOfHotels; i++){
            hotelDistance[i] = scanner.nextInt();
            hotelRating[i] = Double.parseDouble(scanner.next());
        }
        //___calculate-Hotels___//
        calculateNextHotel(new int[5],1,0);
        if(bestHotels != null) {
            //___Print-out-results
            System.out.println("These hotels are the bests:");
            for (int i = 1; i < 5; i++) {
                System.out.println(bestHotels[i] + ". Hotel at: " +
hotelDistance[bestHotels[i]] + " min  Rating: " +
hotelRating[bestHotels[i]]);
            }
            System.out.println("Lowest-Rating: " + bestHotelRating);
        } else {
            System.out.println("There is no possible route");
        }
    }
```

```
    private static void calculateNextHotel(int[] hotelIndexes, int day, int
pastTime){
        //___Check if day is valid___//
        if(day < 5){
            //___Check for each hotel
            for(int i = (hotelIndexes[day-1] == 0) ? 0 : hotelIndexes[day-
1]+1; i < numberOfHotels; i++){
                hotelIndexes[day] = i;
                //___Check if hotelDistance isn't too far away
                if(hotelDistance[i] > pastTime+dailyTravelTime){
                    break;
                //___Check if destination can be reached and if hotel
rating is better than bestHotelRating___//
                } else if(hotelDistance[i]+dailyTravelTime*(5-day) <
totalTravelTime || hotelRating[i] < bestHotelRating){
                    continue;
                }
                //___Calculate next Hotel___//
                calculateNextHotel(hotelIndexes,day+1, hotelDistance[i]);
            }
        } else{
            //___Check if destination is reached___//
            if(pastTime+dailyTravelTime >= totalTravelTime){
                //___Calculate Minimal Rating___//
                double minimalRating =
calculateMinimalRating(hotelIndexes,0);

                //___Check if this is a better minimal Rating
                if(bestHotelRating <= minimalRating){
                    //___Check if it's the same rating then the best___//
                    if(bestHotelRating == minimalRating) {
                        //___Check the second, third and fourth worse
rating is better...//
                        for (int i = 1; i < 4; i++) {
                            //___Calculate (second or third, fourth)
Minimal Rating___//
                            double min1 =
calculateMinimalRating(hotelIndexes, i);
                            //___Calculate (second or third, fourth)
Minimal Rating of Best___//
                            double min2 =
calculateMinimalRating(bestHotels, i);
                            //___Check if it's better than Best___//
                            if (min1 > min2) {
                                break;
                            //___Check if it's worse than Best___//
                            } else if (min1 < min2) {
                                return;
                            }
                        }
                    }
                    //___Set current minimalRating to best___//
                    bestHotelRating = minimalRating;
                    //___Copy hotels and set them to best___//
                    int[] copy = hotelIndexes.clone();
                    bestHotels = copy;
                }
            }
        }
    }
```

```
      private static double calculateMinimalRating(int[] hotelIndexes, int
index){
        //___Create Array with all ratings of the hotels___//
        double[] ratings = new double[hotelIndexes.length-1];
        for(int i = 1; i < hotelIndexes.length; i++){
            ratings[i-1] = hotelRating[hotelIndexes[i]];
        }
        //___Sort this Array using insertion Sort___//
        bubbleSort(ratings);

        //___return rating from index (0 = Minimum, 1 = second minimum)
        return ratings[index];
    }
    private static void bubbleSort(double[] list) {
        //___Create variable Cache for swap___//
        double Cache;
        //___For each element go through the list__//
        for (int i = 0; i < list.length; i++) {
            for (int j = 0; j < list.length - 1 - i; j++) {
                //___Check If Element is greater than the other___//
                if (list[j] > list[j + 1]) {
                    //___Swap-Elements___//
                    Cache = list[j + 1];
                    list[j + 1] = list[j];
                    list[j] = Cache;
                }
            }
        }
    }
}
```