BWINF 40: 1. Runde
Aufgabe 2 : Vollgeladen

Lennart Zeppenfeld

Inhaltsverzeichnis

1 Lösungsidee

Die Aufgabe fordert die beste Route aus einer Liste an Hotels zu finden, bei welcher die niedrigste Bewertung eines Hotels so hoch wie möglich sein muss. Um das zu schaffen war es meine Idee jede mögliche Route zu finden und diese mit der aktuell besten Route zu vergleichen. War sie besser als die aktuell beste Route, so wurde sie die neue beste Route.

Damit ich jede mögliche Route erwische, werde ich mit mehreren Schleifen durch die Hotelliste gehen und laufend prüfen, ob diese Route nach den Regeln der Aufgabe gefahren werden kann.
Eine Route ist besser als eine ander Route, wenn sie eine höhere niedrigste Bewertung eines Hotels hat oder diese gleich ist, aber die durchschnittliche Hotelbewertung besser ist.

Es soll nicht mehr nach weiteren Routen gesucht werden, wenn das erste Mal das erste Hotel einer Route weiter als 360 Minuten entfernt ist. Somit wäre es nicht mehr erreichbar, was bedeutet, dass es auch keine weiteren möglichen Routen mehr gibt.

2 Umsetzung

2.1 DataInput

Zunächst habe ich mich damit beschäftigt einen File Reader in meinem Java Projekt zu implementieren. Damit mein Projekt gut strukturiert ist habe ich für den File Reader die Klasse „DataInput“ erstellt. Diese kümmert sich um das Einlesen und die Verwertung der Daten. Dabei werden Anzahl der Hotels und die Reisedauer bis zum Ziel in der Beispieldatei identifiziert und gespeichert. Darüber hinaus wird hier auch eine Liste von Hotels erstellt.

2.2 Hotel

Um Hotels vernünftig darstellen und deren Daten einfach abspeichern zu können, habe ich eine eigene Klasse „Hotel“ erstellt. Diese besitzt die wichtigsten Eigenschaften, wie z.B. die Bewertung und die Reisedauer zum Hotel.
Die Klasse Hotel wird beim Erstellen der Hotelliste der Klasse „DataInput“ verwendet.

2.3 Route

Als nächsten Schritt habe ich die Klasse „Route erstellt“. Da diese im späteren Hauptteil des Programms mit anderen Routen verglichen werden muss, enthält sie Eigenschaften wie die durchschnittliche Hotelbwertung und die niedrigste Bewertung eines Hotels der Route.

2.4 RoutePlanner

Die Klasse RoutePlanner enthält das Herzstück des Programms. Hier wird die beste Route gefunden und am Ende ausgegeben.
Der RoutePlanner beinhaltet zwei Objekte vom Typ „Route“. Eine sogenannten „tryRoute“ und eine „bestRoute“.
Die Funktion „tryAllRoutes“ geht alle Routen durch und überprüft dabei laufend, ob diese das gegebene Reiseziel erreichen können. Ist das der Fall wird die „tryRoute“ mit der „bestRoute“ verglichen und möglicherweise ausgetauscht.
Das Durchlaufen aller Routen erfolgt durch vier for-Schleifen, welche jeweils durch die Hotelliste laufen. In jeder Schleife wird zunächste überprüft, ob das Hotel vom vorherigen Hotel bzw. das erste Hotel vom Start aus erreichbar ist.

2.5 Main

In der main() wird am Anfang das Objekt „dataInput“ erstellt und somit alle wichtigen Daten aus der Beispieldatei erkannt und gespeichert.
Im Anschluss wird der „routePlanner“ erstellt und dessen Funktion „tryAllRoutes“ ausgeführt. Diese Funktion gibt die beste mögliche Route zurück, welche dann mittels Konsolenausgaben dem Benutzer des Programms dargestellt wird.

3 Beispiele

Mein Algorithmus kann alle der vorliegenden Beispielfälle lösen. Um die Effektivität zu überprüfen habe ich verschieden Analysemöglichkeiten implementiert. Dazu gehören die Zeit, die zum Ermitteln der besten Route benötigt wird, sowie auch die Anzahl an möglichen Routen.

Folgend sind alle Ausgaben zu den verschiedenen Beispieldateien aufgelistet.

3.1 Hotels1

```
Anzahl an Hotels: 12
Reiseziel: 1680
-----------Beste Route für hotels1-----------
Anzahl an Stopps: 4
Durchschnittliche Bewertung: 3.175
Niedrigste Bewertung eines Hotels: 2.7
347 | Bewertung: 2.7
687 | Bewertung: 4.4
1007 | Bewertung: 2.8
1360 | Bewertung: 2.8
Zeit zum Berechnen der besten Route: 0.0 Sekunden
Anzahl an möglichen Routen: 8
```

3.2 Hotels2

```
Anzahl an Hotels: 25
Reiseziel: 1737
-----------Beste Route für hotels2-----------
Anzahl an Stopps: 4
Durchschnittliche Bewertung: 3.775
Niedrigste Bewertung eines Hotels: 2.3
341 | Bewertung: 2.3
700 | Bewertung: 3.0
1053 | Bewertung: 4.8
1380 | Bewertung: 5.0
Zeit zum Berechnen der besten Route: 0.001 Sekunden
Anzahl an möglichen Routen: 126
```

3.3 Hotels3

```
Anzahl an Hotels: 500
Reiseziel: 1793
-----------Beste Route für hotels3-----------
Anzahl an Stopps: 4
Durchschnittliche Bewertung: 2.5999999999999996
Niedrigste Bewertung eines Hotels: 0.3
359 | Bewertung: 4.6
717 | Bewertung: 0.3
1076 | Bewertung: 3.8
1433 | Bewertung: 1.7
```

```
Zeit zum Berechnen der besten Route: 0.839 Sekunden
Anzahl an möglichen Routen: 10
```

3.4 Hotels4

```
Anzahl an Hotels: 500
Reiseziel: 1510
-----------Beste Route für hotels4-----------
Anzahl an Stopps: 4
Durchschnittliche Bewertung: 4.775
Niedrigste Bewertung eines Hotels: 4.6
340 | Bewertung: 4.6
676 | Bewertung: 4.6
1032 | Bewertung: 4.9
1301 | Bewertung: 5.0
Zeit zum Berechnen der besten Route: 1.243 Sekunden
Anzahl an möglichen Routen: 3031821
```

3.5 Hotels5

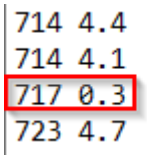
```
Anzahl an Hotels: 1500
Reiseziel: 1616
-----------Beste Route für hotels5-----------
Anzahl an Stopps: 4
Durchschnittliche Bewertung: 5.0
Niedrigste Bewertung eines Hotels: 5.0
280 | Bewertung: 5.0
636 | Bewertung: 5.0
987 | Bewertung: 5.0
1271 | Bewertung: 5.0
Zeit zum Berechnen der besten Route: 105.43 Sekunden
Anzahl an möglichen Routen: 38451283
```

3.6 Fazit

Besonders auffällig sind zwei Gegebenheiten bei den Beispieldaten.
Wenn man bedenkt, dass die Anzahl der Hotels bei den Beispielen 3&4 jeweils bei 500 liegt, ist es ein ungewöhnlich hoher Unterschied bei der Anzahl der möglichen Routen. Bei hotels3 konnten nur 10 Routen das Ziel erreichen, bei hotels4 schafften es jedoch 3.031.821 Routen.
Die kürzere Laufzeit von hotels3 im Vergleich zu hotels4 lässt sich damit erklären, dass mehr als 3 Millionen Routen weniger mit der besten Route verglichen werden mussten.

Des Weiteren ist die viel längere Laufzeit bei hotels5 (105.43s) auffallend. Diese kommt wahrscheinlich von der hohen Anzahl an möglichen Routen (38.451.283) zustande.

Bei hotels3 ist es auch ungewöhnlich, dass die niedrigste Bewertung bei 0.3 liegt, obwohl es insgesamt 500 Hotels gibt. Fast alle Hotels weisen dabei eine höhere Bewertung auf.
Erklären lässt sich die niedrigste Bewertung dieser Route damit, dass pro Reisetag mindestens 358,6 (Reisedauer in min / Reisetage in Tage) Minuten gefahren werden muss.
Betrachtet man nun die entsprechende Stelle des Hotels in der Liste von hotels3, fällt auf, dass es durchaus bessere Hotels in unmittelbarer Nähe gibt.
Allerdings kann das nachfolgende Hotel nicht erreicht werden und die beiden vorliegenden Hotels verhindern eine mögliche Route, in dem die im Nachgang zu erreichenden Hotels es nicht ermöglichen das Ziel zu erreichen.
Somit ist die niedrige Bewertung des Hotels von 0.3 zwar ungewöhnlich, aber nicht vermeidbar.



Beachtlich ist auch die Laufzeit des Algorithmus bei den Beispielen hotels1 (0.0s) & hotels2 (0.01s). Diese ist offensichtlich der niedrigen Anzahl der Hotels und der daraus resultierenden niedrigen Anzahl an möglichen Routen geschuldet.

Grundsätzlich kann man sagen, dass mit einer höheren Anzahl an Hotels auch eine längere Laufzeit zu erwarten ist. Allerdings sagt die Anzahl der Hotels nichts über die Anzahl der möglichen Routen aus (siehe hotels3 & hotels4)

4 Quellcode

4.1 Hotel

```
public class Hotel {
	int timeToTravel;
	double rating;
	boolean isFilled;
	public Hotel(int parTimeToTravel, double parRating, boolean parIsFilled) {
		this.timeToTravel = parTimeToTravel;
		this.rating = parRating;
		this.isFilled = parIsFilled;
	}
}
```

Die Klasse Hotel ist die kleinste Klasse meines Projekts. Hier wird lediglich beim Erstellen des Objektes durch den Konstruktor die Eigenschaften timeToTravel, rating und isFilled zugewiesen.
Der boolean isFilled wird genutzt, um zu erkennen, ob es sich um einen Platzhalter in der Route mit den Werten (0, 0.0, false) oder um ein tatsächliches Hotel handelt.

4.2 Route

Route enthält folgende Eigenschaften und Elemente: Anzahl der Stopps, Liste der Hotels der Route, Durchschnittsbewertung und niedrigste Bewertung.
Die wichtigsten Methoden der Klasse Route sind CalcLowestRating() und CalcAverageRating.

4.2.1 CalcLowestRating()

Diese Methode ermittelt die niedrigste Bewertung eines Hotels in der entsprechenden Route.
Sie beginnt damit die Bewertung des ersten Hotels als niedrigste Bewertung anzusehen. Anschließend wird eine for-Schleife durchlaufen, welche ab dem zweiten Hotel der Route überprüft, ob die Bewertung niedriger als die aktuell niedrigste Bewertung ist.
Die for-Schleife wird so lange durchlaufen, wie Hotels in der Liste der Route vorhanden sind.

```
public void CalcLowestRating () {
	lowestRating = hotelsOfRoute.get(0).rating;
	for(int i = 1; i < hotelsOfRoute.size(); i++) {
		if(hotelsOfRoute.get(i).isFilled == true) {
			if(hotelsOfRoute.get(i).rating < lowestRating) {
				lowestRating = hotelsOfRoute.get(i).rating;
			}
		}
	}
}
```

4.2.2 CalcAverageRating()

Hier wird die durchschnittliche Hotelbewertung der Route berechnet. Dazu werden erst alle Bewertungen zusammengerechnet. Anschließend wird die Anzahl an Hotels in der Route berechnet. Ist diese ungleich 0 kann die Summe der Bewertungen durch die Anzahl an Hotels dividiert werden.

```
public void CalcAverageRating () {
	double allRating = 0;
	for (int i = 0; i < hotelsOfRoute.size(); i++) {
		if (hotelsOfRoute.get(i).isFilled == true) {
			allRating += hotelsOfRoute.get(i).rating;
		}
	}
	CalcAmountOfStops();
	if (amountOfStops != 0) {
		averageRating = allRating / amountOfStops;
	} else {
		averageRating = 0;
	}
}
```

4.3 RoutePlanner

Der wichtigste Teil meines Programms ist die Klasse RoutePlanner. In dieser wird der Algorithmus durchlaufen, welcher die beste mögliche Route ermittelt. Hinzu kommen noch weitere Methoden wie canReachGoal, canReachNextHotel und updateBestRoute.

4.3.1 canReachNextHotel()

```
public boolean canReachNextHotel (int hotelNo1, int hotelNo2) {
	boolean canReach = false;
	Hotel hotel1 = listOfHotels.get(hotelNo1);
	Hotel hotel2 = listOfHotels.get(hotelNo2);
	if (hotel2.timeToTravel - hotel1.timeToTravel <= 360 ) {
		canReach = true;
	}
	return canReach;
}
```

Bei dieser Methode wird überprüft, ob der Abstand zwischen zwei Hotels niedriger oder gleich 360 ist. Das heißt man schaut nach, ob man von Hotel A das Hotel B innerhalb eines Reisetags erreichen kann. Wenn das der Fall ist, gibt die Methode true zurück.

4.3.2 updateBestRoute()

```
public void updateBestRoute (Route tryRoute) {
		amountOfPossibleRoutes++;
		tryRoute.CalcAverageRating();
		tryRoute.CalcLowestRating();

		if(tryRoute.lowestRating > bestRoute.lowestRating) {
			bestRoute.hotelsOfRoute = (ArrayList<Hotel>) tryRoute.hotelsOfRoute.clone();
			bestRoute.CalcAverageRating();
			bestRoute.CalcLowestRating();
		} else if (tryRoute.lowestRating == bestRoute.lowestRating && tryRoute.averageRating > bestRoute.averageRating) {
			bestRoute.hotelsOfRoute = (ArrayList<Hotel>) tryRoute.hotelsOfRoute.clone();
			bestRoute.CalcAverageRating();
			bestRoute.CalcLowestRating();
		}

}
```

Mit dieser Methode wird eine mögliche Route mit der aktuell besten Route verglichen. Dabei wird zunächst geschaut, ob es eine höhere niedrigste Bewertung eines Hotels gibt.
Ist das nicht der Fall wird noch überprüft, ob die niedrigste Bewertung eines Hotels der beiden Routen gleich ist, aber die durchschnittliche Hotelbewertung der neuen möglichen Route höher ist. Trifft dies zu, wird sie zur neuen besten Route.

4.3.3 tryAllRoutes()

```
39          for (int i = 0; i < listOfHotels.size(); i++) {
40              if(listOfHotels.get(i).timeToTravel <= 360) {
41                  tryRoute.hotelsOfRoute.set(0, listOfHotels.get(i));
42                  if (canReachGoal(tryRoute.hotelsOfRoute.get(0).timeToTravel)  ) {
43                      updateBestRoute(tryRoute);
44                      continue;
45                  }
46              } else {
47                  tryRoute.hotelsOfRoute.set(0, new Hotel(0, 0, false));
48                  break;
49              }
50
51              for(int j = i+1; j < listOfHotels.size(); j++) {
52                  if (canReachNextHotel(i, j) ){
53                      tryRoute.hotelsOfRoute.set(1, listOfHotels.get(j));
54                      if (canReachGoal(tryRoute.hotelsOfRoute.get(1).timeToTravel) ) {
55                          updateBestRoute(tryRoute);
56                          continue;
57                      }
58                  }else {
59                      tryRoute.hotelsOfRoute.set(1, new Hotel(0, 0, false));
60                      break;
61                  }
62
63                  for (int k = j + 1; k < listOfHotels.size(); k++) {
64                      if (canReachNextHotel(j, k) ){
65                          tryRoute.hotelsOfRoute.set(2, listOfHotels.get(k));
66                          if (canReachGoal(tryRoute.hotelsOfRoute.get(2).timeToTravel) ) {
67                              updateBestRoute(tryRoute);
68                              continue;
69                          }
70                      } else {
71                          tryRoute.hotelsOfRoute.set(2, new Hotel(0, 0, false));
72                          break;
73                      }
74
75                      for (int l = k+1; l < listOfHotels.size(); l++) {
76                          if (canReachNextHotel(k, l) ){
77                              tryRoute.hotelsOfRoute.set(3, listOfHotels.get(l));
78                              if (canReachGoal(tryRoute.hotelsOfRoute.get(3).timeToTravel)) {
79                                  updateBestRoute(tryRoute);
80                                  continue;
81                              }
82                          } else {
83                              tryRoute.hotelsOfRoute.set(3, new Hotel(0, 0, false));
84                              break;
85                          }
86                      }
87                  }
88              }
89          }
```

Die Methode tryAllRoutes ist das Herzstück des Programms. Sie ermittelt die bestmögliche Route. In dem obigen Ausschnitt sind die vier ineinander verschachtelte for-Schleifen zu erkennen. Es sind vier, da bei fünf Reisetagen maximal in vier Hotels übernachtet werden kann.
In der ersten for-Schleife (Z. 39-49) werden alle Hotels durchgelaufen, solange sie die maximale Reisedauer von 360 Minuten nicht überschreiten. Wird sie überschritten, so greift das else und mittels break (Z. 48) wird der Schleifendurchlauf abgebrochen. Das bedeutet somit, dass keine weitere mögliche Route existieren kann.

Wenn die erste Schleife ein Hotel in Reichweite findet, wird dieses in das Objekt Route auf die erste Listenposition gestellt. Im Anschluss wird überprüft, ob es bereits möglich ist das Ziel vom ersten Hotel aus

zu erreichen (Z. 42). Ist das der Fall wird die Route mit der aktuell besten Route verglichen (Z. 43) und wenn sie besser ist, wird diese zur neuen besten Route. Im Anschluss geht es in der ersten for-Schleife mit dem nächsten Hotel weiter (Z. 44).

Wenn das Ziel noch nicht erreicht werden kann, geht es in der zweiten for-Schleife weiter. Dort wird das nächste Hotel, also mindestens ein Hotel nach dem Hotel auf dem Hotellistenplatz 1 der Route, ausgewählt (Z. 51). Dann wird überprüft, ob man von Hotellistenplatz 1 das neue Hotel überhaupt erreichen kann. Es wird also mit der Funktion canReachNextHotel() (Z. 52) überprüft, ob die Entfernung der beiden Hotels maximal bei 360 liegt. Ist es nicht der Fall, wird der Listenplatz geleert (Z. 59) und mittels break (Z. 60) geht es bei der ersten for-Schleife mit einem neuen Hotel weiter.
Ist es doch der Fall wird das Hotel auf Hotellistenplatz 2 der Route gestellt (Z. 53). Daraufhin wird wieder überprüft, ob man bereits das Reiseziel erreichen kann (Z. 54). Wenn ja wird die Route mit der besten Route verglichen (Z. 55), möglicherweise wird sie dann die neue beste Route und es wird mit dem nächsten Hotel in der zweiten for-Schleife fortgefahren.

Das Prozedere aus der zweiten for-Schleife wiederholt sich in der dritten (Z. 63 – 73) und vierten (Z. 75 – 85). Wenn alle möglichen Routen durchlaufen und überprüft wurden, endet das Programm. Das ist der Fall, wenn es keine zu erreichenden Hotels in der ersten for-Schleife mehr gibt.