Aufgabe 2: Vollgeladen

Team-ID: 00501

Team: MWG go brrrrr

Bearbeiter/-innen dieser Aufgabe:
Theodor Teslia

23. Oktober 2021

Inhaltsverzeichnis

Lösungsidee

Zum Lösen dieser Aufgabe habe ich mich am Auswahlkriterium der Hotels orientiert, nachdem die schlechteste Bewertung möglichst hoch sein soll.

Demnach funktioniert das Programm so, dass es von der besten bis zu schlechtesten Bewertung überprüft, ob es für den Wert eine mögliche Route gibt. Es gibt also eine Schwelle, welche von keiner Bewertung nach unten hin überschritten werden darf. Diese Schwelle startet bei dem Wert 5,0 und wird nach jedem Missglücktem Versuch eine Route zu bilden um 0,1 runtergesetzt.

Um für eine gegebene Schwelle eine Route zu finden, wird ab dem zuletzt benutzten Hotel das nächste Hotel benutzt, dessen Bewertung größer oder gleich der Schwelle ist und das sich so weit weg wie möglich befindet. Dadurch werden die wenigsten Hotels benutzt, welche aber alle gleich oder größer als die Schwelle sind.

Dadurch, dass der Schwellenwert langsam und Schrittweise verringert wird, wird versichert, dass einfach nur eine Route gefunden wird, welche die Mindestbewertung maximiert und dabei ignoriert, welche Werte die anderen Hotelbewertungen haben.

Umsetzung

Implementiert wurde dies, indem mit einer Schleife der Schwellenwert immer weiter verringert und der Algorithmus für diesen Wert durchgeführt wurde. Diese Schleife überprüft nach jedem Durchlauf dieses Algorithmus, ob die Distanz zwischen Ziel der Fahrt und des zuletzt benutzten Hotels kleiner oder gleich der Strecke ist, welche an einem tag zurückgelegt werden kann (360) und

ob höchstens nur vier Hotels benutzt wurden. Dadurch wird die Schleife gestoppt sobald es eine mögliche Route gibt, wodurch der höchste Schwellenwert direkt ermittelt werden kann.

Der Algorithmus funktioniert so, dass linear durch alle Hotels des Beispiels gelaufen wird und für jedes überprüft wird, ob es eine Bewertung besitzt die gleich oder höher als die Schwelle ist. Ist dies nicht der Fall wird einfach mit dem nächsten Hotel der Liste fortgefahren. Danach findet die Überprüfung statt, ob die Distanz zwischen dem Hotel und dem zuletzt benutzten Hotel, bzw. Start geringer als 361, damit das Hotel an einem Tag erreicht werden kann. Wenn dies der Fall ist, wird das Hotel an eine Liste angefügt. Jedes weitere Hotel, welches diese Kriterien erfüllt und auch von dem vorher benutzten Hotel erreicht werden kann ersetzt dann das soeben eingefügte. Somit wird jeden Tag so weit wie möglich gefahren.

Wenn dann das zu überprüfende Hotel zu weit von dem Hotel ist, welches sich nun an der zweitletzten Stelle befindet – es somit also nicht mehr an einem Tag erreicht werden kann – wird das zuletzt eingefügte Hotel „eingeloggt“ und die Überprüfung für das Hotel wird wiederholt.

Der Prozess lässt sich so beschreiben, dass jeder Tag als Zyklus gezählt wird. In jedem Zyklus wird das letzte Element der Liste bearbeitet und immer weiter ersetzt, bis das letzte erreichbare Hotel die Position des letzten Hotels der Liste einnimmt. Dieses wird dann „eingeloggt“ und als neue Basis zur Berechnung der Distanzen genommen.

Unabhängig von dem Algorithmus werden die Bewertungen als Integer Zahlen im Bereich von 1 bis 50 anstatt 0,1 bis 5,0 dargestellt. Dadurch lassen sich die Bewertungen genau vergleichen, da Computer bei Dezimalzahlen mit Komma nicht genau sind.

Ausführen der Datei

Um die beigelegte ausführbare Datei zu testen müssen folgende Schritte befolgt werden.

Die Dateien wurden unter MacOS 11.5.2 erzeugt und mithilfe von folgendem Verfahren getestet.

Im Terminal (`zsh`) muss mithilfe von `cd` das Arbeitsverzeichnis (`pwd`) auf den Ordner bin der Abgabe gesetzt werden. Die Datei lässt sich dann mithilfe von `./Aufgabe2` ausführen. Um verschiedene Beispiele von der BWINF-Webseite zu testen muss einfach nur de Name der Datei, bspw. `hotels4.txt`, mit einem Leerzeichen abgetrennt dahinter geschrieben werden (sofern sich in der gleichen Ebene wie der Ordner `bin` der Ordner `examples` befindet, in dem sich die Beispieldateien unbenannt befinden.

Beispiele

Die Ausgabe des Programmes lässt sich in dem Ordner out finden und wird zudem beim Ausführen der Datei ausgegeben.

Im Folgenden soll nun der Durchlauf des Programmes am Beispiel der Datei hotels1.txt stattfinden.

Der Algorithmus beginnt mit dem Wert 50 als Schwelle (alle Dezimalzeichen wurden entfernt, die Bewertungen wurden alle also mit zehn multipliziert).

Da es in dem Beispiel kein Hotel mit der Bewertung 5,0 gibt, kann dieser Durchlauf, wie auch für 4,9, ignoriert werden. Wenn der Schwellenwert 4,8 beträgt, wird das Hotel bei 326 in die Liste eingefügt. Weiter passiert dann aber auch nichts, da kein anderes Hotel solch eine hohe Bewertung besitzt.

Es wird nun vorgesprungen, bis die Schwelle den Wert 3,0 besitzt:

Das Hotel bei 12 wird in die Liste eingefügt und vom Hotel bei 326 direkt wieder überschrieben, da dieses weiter vorne und trotzdem über dem Schwellenwert ist. Die Hotels bei 347 und 359 werden ignoriert, da deren Bewertungen unter der Schwelle liegen. Da das nächste Hotel bei 553 liegt wird dieser Zyklus abgeschlossen und das Hotel bei 326 wird „eingeloggt“.

Die Bewertung des Hotels bei 553 ist 3,6, weshalb es an die Liste angehängt wird. 590[1] wird aufgrund der Schwelle ignoriert. 687 markiert wieder das Ende des Tages, da von 326 nur bis 686 gefahren werden kann, und 553 wird eingeloggt.

Wieder wird 687 eingefügt, das nächste Hotel ist mit 1007 jedoch zu weit entfernt, wodurch 687 eingeloggt wird.

Die einzigen Hotels des folgenden Zyklus’ sind 1007 und 1008. Beide besitzen aber eine zu geringe Bewertung. Da in diesem Zyklus kein Hotel benutzt werden kann wird der Algorithmus abgebrochen und mit einem verringerten Schwellenwert wieder versucht.

Den ersten erfolgreichen Durchlauf, und dadurch auch die höchste Schwelle, findet man beim Wert 2,7:

Am ersten Tag wird 12 eingefügt, durch 326 und dieses wiederum durch 347 ersetzt. 359 besitzt dagegen eine zu geringe Bewertung. 347 wird also eingeloggt.

Der zweite Tag:

553	590	687
Wird eingefügt	Wird ignoriert	Ersetzt 553 und wird eingeloggt

Der dritte Tag:

1007	1008
Wird eingefügt.	Wird ignoriert. 1007 wird eingeloggt.

Der vierte Tag:

1321	1360
Wird ignoriert	Wird eingefügt und eingeloggt

1 Um den Text zu verkürzen wird folgend nur die drei- bzw. vierstellige Position des Hotels benutzt.

Da es von 1360 bis zum Ende bei 1680 nur 320 Minuten sind, lässt sich dies am fünften Tag fahren und das Ziel wird innerhalb von 5 Tagen mit einer geringsten Bewertung von 2,7 erreicht.

Es folgen nun die Ausgaben für die Beispieldateien:

hotels1.txt:
Die geringste Bewertung eines Hotels ist 2.7
Die Distanzen, an denen ein Hotel aufgesucht wird sind:
347
687
1007
1360

hotels2.txt:
Die geringste Bewertung eines Hotels ist 2.3
Die Distanzen, an denen ein Hotel aufgesucht wird sind:
341
700
1053
1380

hotels3.txt:
Die geringste Bewertung eines Hotels ist 0.3
Die Distanzen, an denen ein Hotel aufgesucht wird sind:
360
717
1076
1433

hotels4.txt:
Die geringste Bewertung eines Hotels ist 4.6
Die Distanzen, an denen ein Hotel aufgesucht wird sind:
340
676
1032
1316

hotels5.txt:
Die geringste Bewertung eines Hotels ist 5
Die Distanzen, an denen ein Hotel aufgesucht wird sind:
317
636
987

1286

Quellcode

```
void makeWay(int ratingThreshold, std::vector<Hotel>& result,
std::vector<Hotel>& list, const int& tripLength) {  // Main algorithm
	int index = 0;  // Marks index in result that gets changed
	bool added = false;
	for (int i = 0; i < list.size(); i++) {
		Hotel h = list.at(i);

		if (h.getScore() < ratingThreshold) {
			continue;
		}
		if (calculateDistance(h.getDistance(), result, index-1) >
maxDistancePerDay) {  // If hotel is to far -> Start new cycle
			if (!added) {
				return;
			}
			index += 1;  // Enter last hotel that was near and good
enough
			added = false;
			i -= 1;  // Try again with same hotel
			if (calculateDistance(tripLength, result) <=
maxDistancePerDay) {
				return;  // Check whether already existing list is
sufficient
			}
			continue;
		}
		if (added) {
			result.at(index) = h;
		} else {  // If first hotel in cycle -> push_back
			result.push_back(h);
			added = true;
		}
	}
}




std::vector<Hotel> result;
int highestPossibleRating = 51;
do {
	if (highestPossibleRating == 0) {
		std::cout << "Es gibt leider keine Möglichkeit innerhalb von
fünf Tagen bzw. vier Übernachtungen nach Portugal zu fahren." << std::endl;
```

```
            if (!printOutFailure(file)) {
                  return -4;
            }
            return 0;
      }
      result.clear();  // Initialize values
      highestPossibleRating -= 1;
      makeWay(highestPossibleRating, result, hotelList, roadtripLength);
} while (calculateDistance(roadtripLength, result) > maxDistancePerDay ||
result.size() >= maxTripLength);
```