Aufgabe 2: Vollgeladen

Team-ID: 00195

Team-Name: WLR

Bearbeiter/-innen dieser Aufgabe:
Baran Peters

21. November 2021

Inhaltsverzeichnis

1 Lösungsidee

Die zu lösende Aufgabe kann man sich wie folgt vorstellen: Man hat ein Baumdiagramm, welches die verschiedenen Auswahlmöglichkeiten der Hotels darstellt. Die Tiefe beträgt hierbei (maximal) 4, wobei jede Ebene für jeweils eine Nacht steht. Anhand dieser Darstellung wird klar, dass es viele verschieden Kombinationen gibt, die innerhalb der gegebenen Zeit von maximal fünf Tagen die verschiedenen Hotels der Reise beschreiben. Ein Teil der theoretisch möglichen Kombinationen fällt allerdings weg, da diese das Ziel nicht in der vorgegebenen maximalen Reisezeit erreichen werden.

Die am Ende in der Aufgabenstellung gegebene Bedingung für die Auswahl der Hotelkombination lässt allerdings Interpretationsspielraum übrig. Man kann hier verstehen, dass Lara und Paul die Kombination mit dem höchsten Durchschnittswert an Hotelbewertungen haben wollen oder, dass die Kombination zum einen einen hohen Durchschnittswert und zum anderen eine kleine Spanne zwischen schlecht und bestbewertestem Hotel haben soll. Beispielsweise wäre dann die Kombination [5,5,5,1] (Zahlenwert = jeweilige Bewertung des Hotels) eigentlich schlechter zu bewerten als [4,4,4,4], da ersteres eine niedrigere Konsistenz der Bewertung aufweist.

Im Folgenden gehe ich für die Lösung der Aufgabe von der ersten Interpretation aus, da diese weniger komplex und zielführender für die Auswahl der Hotels ist.

Zusammengefasst ist der gewählte Lösungsansatz alle möglichen Kombinationen zu sammeln und anhand der vorgegebenen Kriterien die beste Kombination auszuwählen.

Man muss bei diesem Lösungsansatz beachten, dass bei der Eingabe einer hohen Menge an Hotels die Rechenzeit stark ansteigt; die theoretische Menge aller Kombinationen ist stets $n!$, sodass hier mit jedem weiteren Hotel die Kombinationen schneller als eine Exponentialfunktion wachsen. Und für jede einzelne Kombination werden wiederum Rechnungen durchgeführt, welche auch von der Größe des Datensatzes abhängen (s. `filter()` Aufrufe). Das Programm wird daher bei ungefähr $n > 500$ schnell an Rechenzeit zunehmen, sodass beispielsweise kein Ergebnis für $hotels5.txt$ der Beispieldaten berechnet werden kann.

2 Umsetzung

Die Lösungsidee wird in einem Programm der Sprache JavaScript umgesetzt. Voraussetzung ist hierbei die Nutzung der neusten Node.JS Laufzeitumgebung. Nachdem die Daten über die Kommandozeile eingelesen werden, beginnt der hauptsächliche Teil des Programms.

Es werden zuallererst die vom Start an erreichbaren Hotels bestimmt. Über diese iteriert man mithilfe einer for-loop, wobei jede Iteration als Betrachtung eines Asts ausgehend von der ersten Ebene des in der Lösungsidee genannten Baumdiagramms steht. In den jeweiligen Iterationen wird dann die Funktion findNextHotel() aufgerufen, die den rekursiven Algorithmus darstellt, der den Pfad in die nächste Ebene

dieses Diagramms weiterführt. Sie nimmt eine Kombination der Hotels als Parameter, um von dort aus diese Kombination weiterzuführen. Der Algorithmus überprüft zunächst, ob im Sachzusammenhang gesehen überhaupt eine weitere Übernachtung nötig ist. Falls nicht, wird die Kombination in einem Array gespeichert und die Funktion terminiert. Falls ja, wird im Weiteren das nächstmöglich passende Hotel gesucht, das in der Reichweite des Autos an diesem Tag liegt sowie den Zeitpuffer nicht überschreitet, da die Reise sonst in der vorgegebenen Zeitspanne nicht mehr zu beenden ist. Der Zeitpuffer ist die Differenz zwischen der maximalen Ausschöpfung der Reisezeit (360 Minuten mal fünf Tage) und der Zeit in der wir das Ziel erreichen sollen. Falls ein Hotel gefunden wird, ruft die Funktion sich erneut selbst auf mit dem neuen Kombinationspfad. Dadurch wird letztlich jede mögliche Kombination überprüft und erfasst.

Nachdem der Algorithmus alle passenden Kombinationen in einem Array gesammelt hat, wird anschließend das Array nach der durchschnittlichen Bewertung aller Hotels einer Kombination sortiert. Falls es sein sollte, dass zwei gleich "gute" Kombinationen auftreten, wird hier die in der Lösungsidee genannte zweite Interpretation der Bedingung herangezogen und die Option mit der kleineren Spanne beziehungsweise größeren Konsistenz der Bewertungen gewählt.

3 Beispiele

- Ausgabe besteht aus den drei besten Ergebnissen (absteigend). Die einzelnen Ergebnisse, die als Array vorliegen, beinhalten die 4 Hotels in ihrer Reihenfolge, dann ihre durchschnittliche Bewertung sowie die Fließkommazahl zur Bewertung der Konsistenz (s. Lösungsidee)

3.1 Beispieldaten Nr. 1

Eingabe: hotels1.txt aus den Beispiel-Eingaben
Ausgabe:

```
[
  [
    [ 347, '2.7' ],
    [ 687, '4.4' ],
    [ 1007, '2.8' ],
    [ 1360, '2.8' ],
    3.175,
    1.7000000000000002
  ],
  [
    [ 359, '2.6' ],
    [ 687, '4.4' ],
    [ 1007, '2.8' ],
    [ 1360, '2.8' ],
    3.1500000000000004,
    1.8000000000000003
  ],
  [
    [ 347, '2.7' ],
    [ 687, '4.4' ],
    [ 1008, '2.6' ],
    [ 1360, '2.8' ],
    3.125,
    1.8000000000000003
  ]
]
```

3.2 Beispieldaten Nr. 2

Eingabe: hotels2.txt aus den Beispiel-Eingaben
Ausgabe:

```
[
  [
    [ 341, '2.3' ],
    [ 700, '3.0' ],
    [ 1053, '4.8' ],
    [ 1380, '5.0' ],
```

```
    3.775,
    2.7
  ],
  [
    [ 341, '2.2' ],
    [ 700, '3.0' ],
    [ 1053, '4.8' ],
    [ 1380, '5.0' ],
    3.75,
    2.8
  ],
  [
    [ 342, '2.1' ],
    [ 700, '3.0' ],
    [ 1053, '4.8' ],
    [ 1380, '5.0' ],
    3.7249999999999996,
    2.9
  ]
]
```

3.3 Beispieldaten Nr. 3

Eingabe: hotels3.txt aus den Beispiel-Eingaben
Ausgabe:

```
[
  [
    [ 359, '4.6' ],
    [ 717, '0.3' ],
    [ 1076, '3.8' ],
    [ 1433, '1.7' ],
    2.5999999999999996,
    4.3
  ],
  [
    [ 358, '2.5' ],
    [ 717, '0.3' ],
    [ 1076, '3.8' ],
    [ 1433, '1.7' ],
    2.0749999999999997,
    3.5
  ],
  [
    [ 359, '4.6' ],
    [ 717, '0.3' ],
    [ 1075, '0.8' ],
    [ 1433, '1.7' ],
    1.8499999999999999,
    4.3
  ]
]
```

3.4 Beispieldaten Nr. 4

Eingabe: hotels4.txt aus den Beispiel-Eingaben
Ausgabe:

```
[
  [
    [ 255, '4.8' ],
    [ 605, '4.9' ],
    [ 949, '4.5' ],
    [ 1301, '5.0' ],
    4.8,
    0.5
  ],
```

```
  [
    [ 249, '4.8' ],
    [ 605, '4.9' ],
    [ 949, '4.5' ],
    [ 1301, '5.0' ],
    4.8,
    0.5
  ],
  [
    [ 266, '4.7' ],
    [ 605, '4.9' ],
    [ 949, '4.5' ],
    [ 1301, '5.0' ],
    4.775,
    0.5
  ]
]
```

3.5 Beispieldaten Nr. 5

Eingabe: hotels5.txt aus den Beispiel-Eingaben
Ausgabe: OutOfMemory Error ==> Keine Ausgabe
Die in der Lösungsidee angesprochene Schwachstelle des Algorithmus kommt hier zum Vorschein. Wenn man das Programm mit einer großen Hotelanzahl, bei hotels5.txt $n = 1500$, ausführt, wird das Programm nach einer gewissen Zeit abbrechen aufgrund eines OOM-Fehlers, da der Heap, also die vom Betriebssystem zugewiesene Speichermenge, regelrecht überfüllt wurde. Weitere Optimierungsversuche des Algorithmus' wurden leider nicht erfolgreich unternommen. Die .filter Funktion eines Arrays sowie anschließende Iterieren über das Ergebnis konnte mit einer for-loop, die manuell die Bedingung überprüft und direkt fortfährt, ersetzt werden. Zudem habe ich noch den Chrome Profiler herangezogen, der mir die rechenintensivsten Stellen aufzeigte.

```
            // Falls die Bewertungen gleich groß scheinen sollten, wähle den konsistenteren Datensatz aus
            if(diff == 0){
               return a[a.length - 1] - b[b.length - 1]
            } else {
               return diff;
            }
        });

        // Die besten 3 Kombinationen ausgeben (absteigend)
        console.log(combinations.slice(Math.max(combinations.length - 3, 0)).reverse())
    }

    // Rekursiver Algorithmus, der letztlich das nächste mögliche Hotel,
    // ausgehend von dem derzeitigen Stand gegeben durch den Funktionsparameter, berechnet.
106.4 ms function findNextHotel(currentCombinationPath) {  currentCombinationPath = (4) [Array(2), Array(2), Array(2), Array(2)]
        // Zeit, die man schon gefahren ist
  3.6 ms let timeTraveled = currentCombinationPath[currentCombinationPath.length - 1][0]  timeTraveled = 1268, currentCombinationPath = (4) [Array(2), Array(2), Array(2), Array(2)]

        // Hotels, die man erreichen könnte vom jetzigen Stand aus
        let hotelsInReach = hotels.filter(x => x[0] > timeTraveled && x[0] <= timeTraveled + 360)
21634.0 ms

        // Falls die schon gereiste Zeit + die Zeit, die man einem Tag fahren könnte >= die benötigte Zeit ist,
        // wird keine Übernachtung mehr benötigt und der Kombinationspfad als fertig betrachtet.
 54.1 ms if(timeTraveled + 360 >= totalTime){
            // Berechne die durchschnittliche Bewertung sowie den Bereich der Werte, um nicht im Nachhinein ressourcenlastige for-Loops durchzuführen
            let range = [0,5];
 85.3 ms    let avgRating = currentCombinationPath.reduce((p, c) => {
                // Gleichzeitig der Optimierung halber den minimalen sowie maximalen Wert berechnen
2740.6 ms       if(range[0] < +c[1]){
1281.8 ms           range[0] = +c[1]
                }

2544.7 ms       if(range[1] > +c[1]){
1273.2 ms           range[1] = +c[1]
                }


2455.3 ms       return p + +c[1]
 43.1 ms    }, 0) / currentCombinationPath.length
1926.3 ms   combinations.push(currentCombinationPath.concat([avgRating, range[0] - range[1]]))

            // Beenden der Betrachtung weiterer erreichbarer Hotels, da keine Übernachtung mehr benötigt wird.
126.2 ms    return;
        }

111.2 ms for(let x of hotelsInReach){
            // Überprüft ob die maximale Zeit, die man bis zu dem Tag hätte fahren können minus die Zeit, die wir gefahren sind
            // kleiner ist als der im Vorhinein berechnete Puffer...
229.0 ms    if((currentCombinationPath.length + 1) * 360 - (x[0]) <= buffer){
                // ... falls ja, dann starte von diesem Hotel aus erneut den Algorithmus
7681.5 ms       findNextHotel(currentCombinationPath.concat([x]));
                continue;
            } else {
                // ... falls nein, dann beende den Durchlauf dieses Pfads vorzeitig, da das Ziel nicht mehr erreicht werden kann.
 0.5 ms         continue;
            }
```

3.6 Eigenes Beispiel

Eingabe: 15 1680 12 4.3 326 4.8 347 2.7 337 2.5 359 2.6 553 3.6 590 0.8 687 4.4 677 4.4 1007 2.8 990 3.8 1008 2.6 1315 2.1 1360 2.8 1340 1.8 *Ausgabe:*

```
[
  [
    [ 326, '4.8' ],
    [ 677, '4.4' ],
    [ 990, '3.8' ],
    [ 1340, '1.8' ],
    3.7,
    3
  ],
  [
    [ 326, '4.8' ],
    [ 677, '4.4' ],
    [ 1007, '2.8' ],
    [ 1360, '2.8' ],
    3.7,
    2
  ],
  [
    [ 326, '4.8' ],
    [ 677, '4.4' ],
    [ 1008, '2.6' ],
    [ 1360, '2.8' ],
    3.6499999999999995,
    2.1999999999999997
  ]
]
```

Dieses Beispiel zeigt noch einmal auf, dass bei zwei gleich bewerteten Routen die konsistentere Wahl stärker gewichtet wird.

4 Quellcode

```
var hotelNum;
var totalTime;
var buffer;

// Menge aller zur Wahl stehenden Hotels
var hotels = []

// Alle Hotelkombinationen, die zum Ziel führen in der gegebenen Zeit
var combinations = [];

function main() {
    // Anzahl aller Hotels
    hotelNum = +readline();

    // Fahrtzeit in Minuten
    totalTime = +readline();

    // Zeit die wir über haben, falls man in 5 Tagen die maximalen Fahrzeit fährt
    buffer = 360 * 5 - totalTime

    var hotelsInReachOnFirstDay = [];

    for(var i = 0;i<hotelNum;i++){
        // info[0] = Entfernung, info[1] = Bewertung
        let information = readline().split("␣")

        // Umwandlung in int zwecks einfacherer Umrechnung
        information[0] = +information[0]

        // Filtert alle Hotels, die am ersten Tag erreicht werden können & fügt sie einem Array hinzu
        if(information[0] <= 360){
            hotelsInReachOnFirstDay.push(information)
        }

        hotels.push(information);
```

```
    }


    for(var x of hotelsInReachOnFirstDay){
        let currentCombinationPath = []
        currentCombinationPath.push(x)

        // Startet den Algorithmus, der alle Kombinationen beginnend von einem der am ersten Tag erreichbaren Hotels, berechnet.
       findNextHotel(currentCombinationPath)
    }

    // Hotel mit höherer durchschnittlicher Bewertung wählen
    // ==> bei Gleichstand Kombination mit konsistenteren Bewertungen wählen (Abstand zwischen Maximum & Minimum).
    combinations.sort((a, b) => {
        let diff = a[a.length - 2] - b[b.length - 2]

        // Falls die Bewertungen gleich gro{\ss} scheinen sollten, wähle den konsistenteren Datensatz aus
        if(diff == 0){
           return a[a.length - 1] - b[b.length - 1]
        } else {
            return diff;
        }
    });

    // Die besten 3 Kombinationen ausgeben (absteigend)
    console.log(combinations.slice(Math.max(combinations.length - 3, 0)).reverse())
}

// Rekursiver Algorithmus, der letztlich das nächste mögliche Hotel,
// ausgehend von dem derzeitigen Stand gegeben durch den Funktionsparameter, berechnet.
function findNextHotel(currentCombinationPath) {
    // Zeit, die man schon gefahren ist
    let timeTraveled = currentCombinationPath[currentCombinationPath.length - 1][0]

    // Hotels, die man erreichen könnte vom jetzigen Stand aus
    let hotelsInReach = hotels.filter(x => x[0] > timeTraveled && x[0] <= timeTraveled + 360)

    // Falls die schon gereiste Zeit + die Zeit, die man einem Tag fahren könnte >= die benötigte Zeit ist,
    // wird keine Übernachtung mehr benötigt und der Kombinationspfad als fertig betrachtet.
    if(timeTraveled + 360 >= totalTime){
        // Berechne die durchschnittliche Bewertung sowie den Bereich der Werte, um nicht im Nachhinein ressourcenlastige for-loops durchzuführen
        let range = [0,5];
        let avgRating = currentCombinationPath.reduce((p, c) => {
            // Gleichzeitig der Optimierung halber den minimalen sowie maximalen Wert berechnen
            if(range[0] < +c[1]){
                range[0] = +c[1]
            }

            if(range[1] > +c[1]){
                range[1] = +c[1]
            }


            return p + +c[1]
        }, 0) / currentCombinationPath.length
        combinations.push(currentCombinationPath.concat([avgRating, range[0] - range[1]]))
    )

        // Beenden der Betrachtung weiterer erreichbarer Hotels, da keine Übernachtung mehr benötigt wird.
        return;
    }

    for(let x of hotelsInReach){
```

```
100         // Überprüft ob die maximale Zeit, die man bis zu dem Tag hätte fahren können
    minus die Zeit, die wir gefahren sind
            // kleiner ist als der im Vorhinein berechnete Puffer...
102         if((currentCombinationPath.length + 1) * 360 - (x[0]) <= buffer){
                // ... falls ja, dann starte von diesem Hotel aus erneut den Algorithmus
104             findNextHotel(currentCombinationPath.concat([x]));
                continue;
106         } else {
                // ... falls nein, dann beende den Durchlauf dieses Pfads vorzeitig, da das
    Ziel nicht mehr erreicht werden kann.
108             continue;
            }
110     }
}
```