

Aufgabe 2: Vollgeladen

2.1 Lösungsidee

Ein einfacheres Problem

Um Lara und Paul bei der Auswahl der Hotels zu helfen, versuchen wir zuerst, das Problem in ein leichteres zu überführen. Dazu betrachten wir das folgende Problem:

„Ist es möglich, dass Familie Meier ankommt, wenn alle Hotels – unabhängig von der Bewertung – an der Route benutzt werden dürfen?“

Damit Familie Meier ankommen kann, muss sie eine Auswahl an Hotels treffen. Wir bezeichnen die Hotels, in denen Familie Meier übernachten möchte, als H_1, \dots, H_i . Dabei steht H_1 für das erste Hotel an der Route und H_i für das letzte. Laut Aufgabenstellung darf die Reise maximal 5 Tage dauern. Daher dürfen maximal 4 Hotels zur Übernachtung ausgewählt werden. Natürlich muss Familie Meier aber nicht 4 Nächte in Hotels verbringen, sofern die Reise bereits früher beendet ist. Formal bedeutet dies, dass $i \leq 4$ gelten muss.

Ein Sonderfall tritt auf, wenn unsere Reise weniger als 6h dauert und daher ohne Übernachtung gefahren werden kann. Diesen Fall sollte unser Programm gesondert überprüfen und eine informative Meldung ausgeben. In der folgenden Lösung werden wir diesen Fall daher nicht weiter betrachten.

Damit die Auswahl H_1, \dots, H_i der Hotels gültig ist, muss sie die folgenden Bedingungen erfüllen:

- Es wurden maximal 4 Hotels ausgewählt, also $i \leq 4$.
- Das Hotel H_1 ist vom Start „erreichbar“, also maximal 6h vom Start entfernt.
- Jedes weitere Hotel H_j mit $j \geq 2$ ist vom vorherigen Hotel H_{j-1} in max. 6h erreichbar.
- Vom letzten Hotel H_i ist das Ziel der Reise in höchstens 6h erreichbar.

Um also festzustellen, ob Familie Meier ihr Ziel erreichen kann, müssen wir überprüfen, ob eine solche gültige Auswahl an Hotels aus unserer Liste existiert. Dazu benutzen wir die folgende Beobachtung: „Es ist immer optimal, das am weitesten entfernte noch erreichbare Hotel als nächstes auszuwählen“. Diese Beobachtung ergibt sich intuitiv daraus, dass es nie falsch sein kann, noch mögliche Strecke am aktuellen Tag zurückzulegen, anstatt sie auf den nächsten Tag aufzuschieben – schließlich kann dann am nächsten Tag eventuell noch weiter gefahren werden⁶.

Mit dieser Beobachtung können wir also einen Algorithmus entwickeln, der prüft, ob in einer Liste L an Hotels eine gültige Auswahl existiert. Dazu nehmen wir vorerst an, dass die Liste L aufsteigend sortiert nach Position der Hotels ist – d.h. das erste Hotel an der Route steht auch an erster Stelle der Liste. Aus dieser Liste wählen wir nun das am weitesten noch vom Start erreichbare Hotel aus (dieses wird H_1). Im nächsten Schritt wählen wird dann das am weitesten entfernte, noch von H_1 erreichbare Hotel (dieses wird H_2). Diesen Prozess wiederholen wir, bis wir das Ziel erreicht haben. In diesem Fall gibt unser Algorithmus *true* zurück.

Natürlich kann es aber auch passieren, dass wir das Ziel nicht erreichen können. Zum Beispiel ist es möglich, dass kein Hotel von unserem aktuellen Punkt mehr erreicht werden kann, oder

⁶Angelehnt an das bekannte Sprichwort „Was du heute kannst besorgen, das verschiebe nicht auf morgen.“

dass Familie Meier das Ziel nicht mit 4 Übernachtungen erreichen kann. In einem solchen Fall gibt unser Algorithmus den Wert *false* zurück.

Im folgenden Pseudocode nehmen wir an, dass die Variable *n* die Anzahl der Hotels enthält, und die Position unseres Ziels in der Variable *Ziel* gespeichert ist. Unser Algorithmus sieht dann wie folgt aus, wobei *j* den Index des am weitesten entfernten, aber immer noch erreichbaren Hotels darstellt:

Algorithmus 2 Lösung des einfacheren Problems

```

1: procedure ZIELERREICHBAR(L)
2:   letzte_position  $\leftarrow 0$ 
3:   j  $\leftarrow 1$ 
4:   if hotel_position(L[1])  $> 6h$  then
5:     return false                                 $\triangleright$  Das erste Hotel ist nicht vom Start erreichbar
6:   end if
7:   for i  $\leftarrow 1, \dots, 4$  do
8:     while j  $< n$  and hotel_position(L[j + 1])  $- \text{letzte\_position} \leq 6h$  do
9:       j  $\leftarrow j + 1$ 
10:    end while
11:    Hi  $\leftarrow L[j]$                        $\triangleright H_i$  ist das am weitesten entfernte erreichbare Hotel.
12:    letzte_position  $\leftarrow \text{hotel\_position}(H_i)$ 
13:    if Ziel  $- \text{letzte\_position} \leq 6h$  then
14:      return true                             $\triangleright$  Das Ziel ist vom aktuellen Hotel erreichbar.
15:    end if
16:  end for
17:  return false                            $\triangleright$  Das Ziel ist nicht mit 4 Übernachtungen erreichbar.
18: end procedure

```

Für eine Laufzeitanalyse beobachten wir, dass in der *innersten* while-Schleife *j* immer um eins erhöht wird. Da die Schleife aber sofort beendet wird, sobald *j* $> n$, kann diese Schleife höchstens *n*-mal ausgeführt werden. Da die äußere Schleife nur konstant oft durchlaufen wird, ergibt sich damit insgesamt eine Laufzeit von $\mathcal{O}(n)$.

Filtern der Hotels Nun, da wir eine Lösung für das obige Problem gefunden haben, wollen wir nun auch die Bewertungen der Hotels in unser Programm einfließen lassen. Dazu stellen wir uns die Frage „Ist es möglich, dass Familie Meier ankommt, wenn alle Hotels mit mindestens Bewertung *X* an der Route benutzt werden dürfen?“ Eine Bewertung *X*, für welche Familie Meier ihr Ziel erreichen kann, nennen wir im folgenden auch eine *gültige* Bewertung.

Um dieses Problem zu lösen, können wir unseren obigen Algorithmus benutzen. Jedoch schränken wir nun die Liste *L* an verfügbaren Hotels soweit ein, dass nur noch Hotels mit einer Bewertung besser oder gleich *X* auf der Liste auftauchen. Wenn es nun möglich ist, mit den Hotels aus *L* das Ziel zu erreichen, ist *X* also eine gültige Bewertung. Damit wissen wir, dass die Bewertung aller ausgewählten Hotels mindestens *X* sein muss. Insbesondere bedeutet das auch, dass die schlechteste Bewertung eines ausgewählten Hotels mindestens *X* ist.

Finden der optimalen Bewertung

Lineare Suche Nun sind wir bereit, unser originales Problem zu lösen. Dazu müssen wir die beste gültige Bewertung X finden. Um diese zu finden, können wir einfach absteigend über alle möglichen Bewertungen iterieren und überprüfen, ob diese gültig ist. Die erste gültige Bewertung, welche wir in unserer Schleife finden, ist also die höchste und damit unsere gesuchte Lösung.

Insgesamt iteriert unser Algorithmus also im schlimmsten Fall einmal über alle möglichen Bewertungen. Da es nicht mehr mögliche Bewertungen als Hotels geben kann, kann es also höchstens n (wobei n die Anzahl an Hotels ist) Iterationen geben. In jeder Iteration wird überprüft, ob die Bewertung gültig ist. Dazu wird in jeder Iteration die oben beschriebene Methode benutzt, welche $\mathcal{O}(n)$ Zeit benötigt. Insgesamt ergibt sich also eine Laufzeit von $\mathcal{O}(n^2)$.

In allen BWINF-Beispielen tauchen jedoch nur die Bewertungen 0, 0 – 5, 0 auf. Da unter dieser Annahme nur 51 verschiedene Bewertungen möglich sind, ist unsere Laufzeit (bei korrekter Behandlung von doppelten Bewertungen) in der Praxis auf den BWINF-Beispielen etwas schneller als die theoretische Analyse ergeben würde.

Binäre Suche Die Laufzeit zum Finden des optimalen Hotels lässt sich mittels *binärer Suche* noch weiter verbessern. Dazu verwenden wir die folgenden beiden Beobachtungen⁷:

- Wenn die Bewertung X gültig ist, dann ist eine niedrigere Bewertung als X ebenfalls gültig.
- Wenn die Bewertung X nicht gültig ist, dann kann eine höhere Bewertung als X ebenfalls nicht gültig sein.

Wenn wir nun also eine Liste an sortierten, möglichen Bewertungen haben, können wir für den *Median aller Bewertungen* prüfen, ob dieser gültig ist. Dort können die folgenden beiden Fälle auftreten:

1. Der Median ist eine gültige Bewertung: Dann müssen wir für alle niedrigeren Bewertungen nicht mehr prüfen, ob diese gültig sind – da diese garantiert gültig sind.
2. Der Median ist keine gültige Bewertung: Dann müssen wir für alle höheren Bewertungen die Gültigkeit nicht überprüfen, da diese garantiert ungültig sind.

Da für den Median an Bewertungen jeweils die Hälfte aller Bewertungen niedriger oder höher ist, können wir in diesem Schritt die Hälfte aller möglichen Bewertungen ausschließen. Diesen Prozess können wir iterativ fortsetzen, bis nur noch eine Bewertung übrig bleibt. Da wir nach diesem Prozess die Gültigkeit aller Bewertungen kennen, können wir sehr leicht die optimale Bewertung herausfinden.

Unser Verfahren ist sehr effizient, da in jedem Schritt die Hälfte aller restlichen Bewertungen als Lösung ausgeschlossen wird. Damit kann es maximal $\log n$ viele Schritte geben. Weiterhin wird in jedem Schritt überprüft, ob eine Bewertung gültig ist, was $\mathcal{O}(n)$ dauert. Insgesamt ergibt sich somit eine Laufzeit von $\mathcal{O}(n \cdot \log n)$.

⁷Mathematik-Profis werden bemerken, dass beide Beobachtungen äquivalent sind.

2.2 Alternative Lösungsideen

Naives Probieren

Um die Hotels H_1, \dots, H_i zu finden, ist es auch möglich, alle Kombinationen an Hotels durchzuprobieren. Die Überprüfung, ob eine Auswahl gültig ist, kann dann anhand der Kriterien in jeweils konstanter Zeit für feste H_1, \dots, H_i passieren. Da maximal 4 Hotels ausgewählt werden müssen, ergibt sich somit eine (sehr ineffiziente) Laufzeit von $\mathcal{O}(n^4)$. Mit effizientem Code und durch einige Optimierungen (z.B. nur aufsteigende Hotels auswählen) können aber auch mit dieser Methode für alle BWINF-Beispiele Lösungen in adäquater Zeit gefunden werden. Dieser Ansatz ist jedoch trotzdem aufgrund seiner Laufzeit nicht zu empfehlen, und kann auch schon bei etwas mehr Hotels als in den vorgegebenen Beispielen keine Lösung mehr finden.

Dynamische Programmierung

Eine weitere alternative Lösung der Aufgabe ergibt sich mit Hilfe von *dynamischer Programmierung* (DP). Mit zweidimensionaler dynamischer Programmierung kann $dp[i][j]$ berechnet werden. Wir definieren diesen Wert als die optimale Bewertung, die erreicht werden kann, wenn wir die Fahrt an Position i beginnen und noch j Übernachtungen nutzen können.

Unter dieser Definition kann dieser Wert (in Abhängigkeit von anderen zuvor berechneten Werten) folgendermaßen berechnet werden:

$$dp[i][j] = \max_{\substack{\text{Hotel } \ell \text{ an Position } i < k \leq i+6h}} \min(dp[k][j-1], \text{bewertung}(\ell))$$

Aufgepasst werden muss in Randfällen (z.B. $j = 0$). Das Endergebnis steht dann nach der Berechnung in $dp[0][4]$. Die entsprechenden Hotelpositionen können mit dieser Methode ebenfalls berechnet werden, indem zusätzlich zu $dp[i][j]$ auch das dort verwendete Hotel ℓ gespeichert wird.

Der Wert $dp[i][j]$ wird nur benötigt, falls $i = 0$ oder falls es ein Hotel an Position i gibt. j kann nur Zahlen zwischen 0 und 4 annehmen. Somit müssen insgesamt lediglich $5n$ verschiedene DP-Werte berechnet werden. Die Berechnung eines DP-Eintrags $dp[i][j]$ kann mithilfe der Formel oben in Laufzeit $\mathcal{O}(n)$ implementiert werden (indem für jedes Hotel überprüft wird, ob dessen Position zwischen i und $i + 6h$ liegt). Dadurch beträgt die asymptotische Laufzeit insgesamt $\mathcal{O}(n^2)$, aber es gibt auch weitere Möglichkeiten für eine Implementierung mithilfe von DP, mit denen andere Laufzeiten erreicht werden.

2.3 Beispiele

Angewandt auf die Eingaben auf der BWINF-Website liefert das Programm folgende Hotels, die ausgewählt werden können, um eine bestmögliche Bewertung des schlechtesten Hotels zu erreichen. Angegeben ist jeweils die Position des Hotels, sowie die zugehörige Bewertung. Eine korrekte Lösung kann auch andere Hotels auswählen, solange deren minimale Bewertung dieselbe bleibt.

	minimale Bewertung	Hotel 1	Hotel 2	Hotel 3	Hotel 4
hotels1.txt	2.7	347 (2.7)	687 (4.4)	1007 (2.8)	1360 (2.8)
hotels2.txt	2.3	341 (2.3)	700 (3.0)	1053 (4.8)	1380 (5.0)
hotels3.txt	0.3	360 (1.3)	717 (0.3)	1076 (3.8)	1433 (1.7)
hotels4.txt	4.6	340 (4.6)	676 (4.6)	1032 (4.9)	1316 (4.9)
hotels5.txt	5.0	317 (5.0)	636 (5.0)	987 (5.0)	1286 (5.0)

2.4 Bewertungskriterien

Die Bewertungskriterien vom Bewertungsbogen werden hier erläutert (Punktabzug in []).

- [-1] **Lösungsverfahren fehlerhaft**

Der Fall, dass mehrere Hotels unterschiedlicher Bewertung an der gleichen Stelle stehen, muss korrekt behandelt werden. Dies bezieht sich insbesondere auf `hotels2.txt`, dort steht an Stelle 341 ein Hotel mit Bewertung 2.2 und eines mit 2.3. Bei Hotelbewertungen darf davon ausgegangen werden, dass diese im Bereich 0.0 – 5.0 liegen.

- [-1] **Bedingungen nicht korrekt eingehalten**

Entsprechend der Aufgabenstellung dürfen nur ≤ 4 Übernachtungen genutzt werden. Dies bezieht sich insbesondere auf `hotels3.txt`, da dort mit mehr Übernachtungen eine bessere Minimalbewertung erzielt werden kann. Es wird maximal 6h am Tag gefahren.

- [-1] **Verfahren bzw. Implementierung unnötig aufwendig / ineffizient**

Auch auf den größeren Musterbeispielen sollte eine Lösung in angemessener Zeit terminieren. Auch bei „naiven“ $\mathcal{O}(n^4)$ -Lösungen sollte der Output innerhalb weniger Minuten für alle Beispiele generiert werden. Ein konstanter Faktor ist nur problematisch, wenn die Laufzeit bereits in $\mathcal{O}(n^4)$ liegt. Das Verfahren sollte insbesondere keine unnötigen Dinge tun, z. B. alle möglichen Bewertungen von 0,0 bis 5,0 in Schritten von 0,1 durchsuchen, anstatt nur die in der Eingabe vorkommenden Bewertungen.

- [-1] **Ergebnisse schlecht nachvollziehbar**

Die niedrigste Bewertung der ausgewählten Hotels muss klar ersichtlich sein. Die Liste aller ausgewählten Hotels wird ausgegeben.

- [-1] **Beispiele fehlerhaft bzw. zu wenige oder ungeeignete Beispiele**

Die Dokumentation soll Ergebnisse zu mindestens 3 der vorgegebenen Beispiele (`hotels1.txt` bis `hotels5.txt`) enthalten.