

# Aufgabe 1: Schiebeparkplatz

Team-ID: 00067

Team-Name: Panic! at the Kernel

Bearbeiter/-innen dieser Aufgabe:  
Christopher Besch

20. November 2021

## Inhaltsverzeichnis

<b>1</b>	<b>Lösungsidee</b>	<b>1</b>
1.1	Berechnung der Tabellen . . . . .	2
<b>2</b>	<b>Umsetzung und Quellcode</b>	<b>2</b>
<b>3</b>	<b>Beispiele</b>	<b>3</b>

## 1 Lösungsidee

Fuer jedes Hotel laesst sich sagen, ob es innerhalb von  $d$  Tagen erreicht werden kann. Ist dies fuer ein  $d$  der Fall, kann fuer die minimale Bewertung aller auf dem optimalen Weg zu ihm liegenden Hotels angegeben werden. Dieser optimale Weg benoetigt genau  $d$  Tage. Zudem ist er in dem Sinne optimal, dass kein anderer Weg existiert, der in  $d$  Tagen das Hotel erreicht und eine besserer minimale Bewertung aufweist. Diese Werte koennen in einer Tabelle—der Minimumstabelle—dargestellt werden:

		min rating at day:					
idx	rating	0	1	2	3	4	5
===	=====	=	=	=	=	=	=
0	inf	inf					
1	4.3		4.3				
2	4.8		4.8	4.3			
3	2.7		2.7	2.7	2.7		
4	2.6		2.6	2.6	2.6	2.6	
5	3.6			3.6	3.6	2.7	2.6
6	0.8			0.8	0.8	0.8	0.8
7	4.4			2.7	3.6	3.6	2.7
8	2.8				2.7	2.8	2.8
9	2.6				2.6	2.6	2.6
10	2.1					2.1	2.1
11	2.8					2.7	2.8
12	3.3						2.7
13	inf						2.7

Neben den Hotels werde hier mit dem Index 0 der Startort und mit 13 das Ziel gelistet. Dessen Bewertung ist nie schlechter als die eines beliebigen Hotels, weshalb sie als unendlich angesehen wird. Da eine Fahrt, die laenger als fuerf Tage dauert nicht zulaessig ist, werden diese Optionen weggelassen.

Anhand dieser Daten laesst sich recht leicht die minimale Bewertung auf dem Weg zu jedem beliebigen Hotel ausgeben. Interessant ist dieser Wert fuer das Ziel. Es muss lediglich das Minimum aller Werte in der letzten Zeile bestimmt werden.

Um nun die einzelnen Hotels, die optimal zum Ziel fuhren, zu bestimmen, wird eine zweite Tabelle—die Vorgaengertabelle—benoetigt. Diese weist die gleichen Dimensionen auf, gibt allerdings fuer jedes Hotel  $x$  und Tag  $d$  das Hotel, das am vorherigen Tag zuletzt besucht wurde, an. So kann vom Ziel am optimalen Tag ausgehend immer das zuletzt besuchte Hotel bestimmt werden. Dieser Vorgang endet, wenn man am Anfang angekommen ist.

## 1.1 Berechnung der Tabellen

Als Taglaenge wird die Strecke verstanden, die an einem Tag zurueckgelegt werden kann. Zu jedem Hotel  $x$  kann man am Tag  $d$  von allen Hotels, die maximal eine Taglaenge vor  $a$  liegt und innerhalb von genau  $d - 1$  Tagen erreichbar sind, gelangen. Diese Hotels bilden die Menge  $Y$ . Alle Hotels, die hinter  $x$  liegen, sind nicht zu verwenden. Die minimale  $a$  Bewertung zum Hotel  $x$  nach  $d$  Tagen entspricht mit

- dem Minimum  $b$  der Werte aus der Minimumstabelle fuer aller Hotels aus  $Y$  und den Tag  $d - 1$  und
- der Bewertung des Hotels  $x$ :

$$a = b + c$$

$a$  wird in der Minimumstabelle gespeichert. Der entsprechende Wert in der Vorgaengertabelle entspricht dem Index des Hotels dessen

Da so immer nur die Information von Hotels, die vor dem aktuellen liegen, benoetigt werden, kann das Prinzip der dynamischen Programmierung verwendet werden. Hierbei werden die Tabellen anfangend beim „Start“ Hotel fuer Hotel gefuehlt. Jedes weitere Hotel benoetigt ausschliesslich die bereits berechnete Information.

## 2 Umsetzung und Quellcode

Die Loesungsidee wird in C++ implementiert. Zur Repraesentation der Tabellen werden mehrdimensionale vectors benutzt:

```
1 // min_ratings[x][y] -> min rating till hotel x requiring y days to get to
  // -1 -> impossible
3 std::vector<std::vector<float>>> min_ratings(n + 2, std::vector<float>(DAYS + 1, -1));
  std::vector<std::vector<int>>> last_hotel(n + 2, std::vector<int>(DAYS + 1, -1));
```

DAYS entspricht der Anzahl an Tagen, die die Reise maximal dauern darf. Diese Konstante wird an allen relevanten Stellen respektiert, weshalb leicht eine Berechnung fuer mehr (oder weniger) als fuenf Tagen durchgefuehrt werden kann.

Nun kann mit der Funktion `populate_tables` diese Tabellen der Loesungsidee nach fuellen.

```
void populate_tables(
2     const std::vector<std::pair<int, float>>& hotels,
      std::vector<std::vector<float>>& min_ratings,
4     std::vector<std::vector<int>>& last_hotel)
{
6     // the "first" (virtual) hotel never has the worst rating
    min_ratings[0][0] = inf;
8     // go through all but "first" hotels
    for(auto cur = hotels.begin() + 1; cur != hotels.end(); ++cur) {
10         int cur_idx = cur - hotels.begin();
        // first hotel 'cur' can be reached from
12         auto prev = std::lower_bound(hotels.begin(), hotels.end(),
                                         std::make_pair(cur->first - DAY_LEN, .0f));
14         for(; prev != cur; ++prev) {
            int prev_idx = prev - hotels.begin();
16             // go through past days when prev can be reached
            // call ride off after 'DAYS' days
18             for(int prev_day = 0, cur_day = 1; prev_day < DAYS; ++prev_day, ++cur_day) {
                // if the hotel 'prev' can't be reached in 'prev_day' days
20                 if(min_ratings[prev_idx][prev_day] == -1)
                    continue;
22                 float new_min = std::min(min_ratings[prev_idx][prev_day], cur->second);
                // update when better
24                 if(new_min > min_ratings[cur_idx][cur_day]) {
                    min_ratings[cur_idx][cur_day] = new_min;
```

```

26         last_hotel[cur_idx][cur_day] = prev_idx;
27     }
28 }
29 }
30 }
31 }

```

Wenn die Tabellen produziert wurde, koennen mit `get_best_day` die Wert des „Zielhotel“ in der Minimumstabelle linear nach dem Optimum durchsucht werden:

```

int get_best_day(const std::vector<std::vector<float>>& min_ratings, int n)
{
    int best_day {-1};
    float best_min_rating {-1};
    for(int i = 0; i < DAYS + 1; ++i) {
        if(min_ratings[n + 1][i] > best_min_rating) {
            best_min_rating = min_ratings[n + 1][i];
            best_day = i;
        }
    }
    return best_day;
}

```

Wurde festgestellt, dass es einen Weg zum „Zielhotel“ innerhalb von fuenf Tagen gibt, kann der optimale Weg mithilfe der Funktion `construct_path` bestimmt werden:

```

void construct_path(
    const std::vector<std::vector<int>>& last_hotel,
    int best_day,
    int n,
    std::vector<int>& path)
{
    int cur_idx = last_hotel[n + 1][best_day];
    for(int cur_day = best_day - 1; cur_day; --cur_day) {
        path.push_back(cur_idx);
        cur_idx = last_hotel[cur_idx][cur_day];
    }
    std::reverse(path.begin(), path.end());
}

```

Hierbei wird `cur_idx` immer so aktualisiert, dass immer das jeweilig vorhergehende Hotel referenziert wird. Zum Schluss werden die beiden Tabellen und der finale, optimale Weg ausgegeben.

### 3 Beispiele

Das Pythonscript `tabs.py` wird benutzt, um tabs korrekt zu formatieren. Die + Zeichen zeigen, dass dieses Hotel im optimalen Weg enthalten ist.

#### Hotels 1

```

./tabs.py <(cat examples/hotels1.txt | ./a.out)

```

idx	distance	rating	min rating at day:						
			0	1	2	3	4	5	6
0	0	inf							
1	12	4.3		4.3					
2	326	4.8	+	4.8	4.3				
3	347	2.7		2.7	2.7	2.7			
4	359	2.6		2.6	2.6	2.6	2.6		
5	553	3.6	+		3.6	3.6	2.7	2.6	
6	590	0.8			0.8	0.8	0.8	0.8	0.8
7	687	4.4	+		2.7	3.6	3.6	2.7	2.6
8	1007	2.8	+			2.7	2.8	2.8	2.7
9	1008	2.6				2.6	2.6	2.6	2.6
10	1321	2.1					2.1	2.1	2.1

```

16 11      1360      2.8      +
17 12      1411      3.3
18 13      1680      inf
                                     2.7      2.8      2.8
                                     2.7      2.8
                                     2.7      2.8

20 These hotels should be used with min rating 2.8:
   idx      location      rating  min rating till here
22 ===      =====      =====
23 2         326          4.8      4.8
24 5         553          3.6      3.6
25 7         687          4.4      3.6
26 8         1007         2.8      2.8
27 11        1360         2.8      2.8

```

## Hotels 2

```

1 ./tabs.py <(cat examples/hotels2.txt | ./a.out)
                                     min rating at day:
3 idx      distance rating  0      1      2      3      4      5      6
4 ===      =====
5 0         0         inf      inf
6 1         340        1.6
7 2         341        2.2      2.2      1.6
8 3         341        2.3      +      2.3      2.2      1.6
9 4         342        2.1      2.1      2.1      2.1      1.6
10 5         360        1.9      1.9      1.9      1.9      1.6
11 6         361        4.4      2.3      2.2      2.1      1.9      1.6
12 7         362        3.1      2.3      2.3      2.2      2.1      1.9
13 8         442        5         2.3      2.3      2.3      2.2      2.1
14 9         567        4.9      2.3      2.3      2.3      2.3      2.2
15 10        700        3         +      2.3      2.3      2.3      2.3
16 11        710        2.9      1.9      2.3      2.3      2.3      2.3
17 12        718        1.4      1.4      1.4      1.4      1.4      1.4
18 13        987        4.6      2.3      2.3      2.3      2.3      2.3
19 14        1051       2.3      +      2.3      2.3      2.3      2.3
20 15        1053       4.8      2.3      2.3      2.3      2.3      2.3
21 16        1057       0.2      0.2      0.2      0.2      0.2      0.2
22 17        1199       5         2.3      2.3      2.3      2.3
23 18        1279       5         2.3      2.3      2.3      2.3
24 19        1367       4.5      2.3      2.3      2.3      2.3
25 20        1377       1.8      1.8      1.8      1.8      1.8
26 21        1377       1.6      1.6      1.6      1.6      1.6
27 22        1377       2         2         2         2
28 23        1378       2.1      2.1      2.1      2.1
29 24        1378       2.2      2.2      2.2      2.2
30 25        1380       5         +      2.3      2.3      2.3      2.3
31 26        1737       inf      2.3      2.3      2.3

33 These hotels should be used with min rating 2.3:
   idx      location      rating  min rating till here
35 ===      =====
36 3         341          2.3      2.3
37 10        700          3         2.3
38 14        1051         2.3      2.3
39 25        1380         5         2.3

```

## Hotels 3

```

1 ./tabs.py <(cat examples/hotels3.txt | ./a.out)
these hotels should be used with min rating 0.3:
3 idx      location      rating  min rating till here
4 ===      =====
5 97        358          2.5      2.5
6 196       717          0.3      0.3
7 297       1075         0.8      0.3
8 401       1433         1.7      0.3

```

Aufgrund von 500 Hotels wird fuer dieses Beispiel die Minimumstabelle nicht abgedruckt.

## Hotels 5

```
./tabs.py <(cat examples/hotels5.txt | ./a.out)
2 These hotels should be used with min rating 5:
  idx      location      rating  min rating till here
4 ===      =====
  242      280           5        5
6 581      636           5        5
  913      987           5        5
8 1168     1271          5        5
```

Hier wird die Minimumstabelle ebenfalls weggelassen.

## Unmoegliches Beispiel

```
./tabs.py <(cat examples/my_hotels0.txt | ./a.out)
2 min rating at day:
  idx      distance rating  0    1    2    3    4    5    6
4 ===      =====
  0         0         inf  inf
6 1        1680        inf

8 It is impossible to reach the destination in 6 days without draining the phones' batteries.
  Maybe read a book instead.
```

Dieses Beispiel enthaelt kein einziges Hotel, weshalb die gesamte Fahrzeit von 1680 Minuten nicht ohne geladene Smartphones ueberstanden werden kann.

## Moegliches Beispiel nach sechs Tagen

```
1 ./tabs.py <(cat examples/my_hotels1.txt | ./a.out)
  min rating at day:
3 idx      distance rating  0    1    2    3    4    5
  ===      =====
5 0         0         inf  inf
  1         300         5      5
7 2         600         5      5
  3         900         5      5
9 4        1200         5      5
  5        1500         5      5
11 6        1800        inf

13 It is impossible to reach the destination in 5 days without draining the phones' batteries.
  Maybe read a book instead.
```

Es zeigt sich, dass die Hotels geradeso nicht ausreichen, um das Ziel innerhalb von fuenf Tagen zu erreichen. Wenn allerdings die Konstante DAYS auf 6 gesetzt wird, ist das Beispiel loesbar:

```
./tabs.py <(cat examples/my_hotels1.txt | ./a.out)
2 min rating at day:
  idx      distance rating  0    1    2    3    4    5    6
4 ===      =====
  0         0         inf  inf
6 1         300         5      +    5
  2         600         5      +    5
8 3         900         5      +    5
  4        1200         5      +    5
10 5        1500         5      +    5
  6        1800        inf      5

12
  These hotels should be used with min rating 5:
14 idx      location      rating  min rating till here
  ===      =====
```

<sup>16</sup>	1	300	5	5
	2	600	5	5
<sup>18</sup>	3	900	5	5
	4	1200	5	5
<sup>20</sup>	5	1500	5	5