technische universität dortmund

Fakultät für Informatik
Lehrstühle 11, 13, 14

Jun.-Prof. Dr. Maike Buchin
Łukasz Czajka, N. Kriege, D. Kurz, M. Ludewig

Sommersemester 2019
25. Juni

DAP2 Praktikum – Blatt 14

Abgabe: 8.–12. Juli

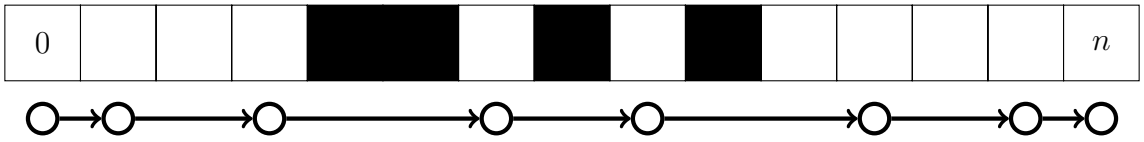
Wichtig: Der Quellcode ist natürlich mit sinnvollen Kommentaren zu versehen. Überlegen Sie außerdem, in welchen Bereichen Invarianten gelten müssen, und überprüfen Sie diese ggf. an sinnvollen Stellen mit *Assertions* (siehe Hinweis auf Blatt 2).

Langaufgabe 14.1: Hindernislauf (8 Punkte)

Gegeben ist ein eindimensionales Feld der Länge $n+1$ (ein *Parcours*) und eine Zahl $r \in \mathbb{N}$. Auf der Zelle 0 des Parcours steht ein Roboter, der sich mit Rechtsschritten auf diesem Parcours bewegen kann und der die Zelle n mit möglichst wenigen Schritten erreichen soll.

In jedem Schritt kann der Roboter seine Schrittweite anpassen, allerdings immer nur um eins nach oben oder unten. Dabei darf die maximale Schrittweite r nicht überschritten werden und der Roboter darf nicht stehen bleiben (also Schrittweite 0 haben), bevor er nicht die Zelle n erreicht hat. Damit der Roboter am Ende auf dieser Zelle stehen bleiben kann, muss er mit der Schrittweite 1 dort ankommen; der letzte Schritt von Zelle n nach Zelle n mit Schrittweite 0 soll dabei nicht mitgezählt werden. Initial wartet der Roboter auf der Startzelle 0; dies entspricht einer Schrittweite von 0, so dass sein erster Schritt von Zelle 0 aus nur Schrittweite 1 haben darf. Zusätzlich befinden sich auf dem Parcours verteilt Hindernisse, die mit einer geeigneten Schrittweite übersprungen werden müssen, damit der Roboter sein Ziel erreicht. Der Roboter darf sich nur in positiver Richtung bewegen.

Ein Beispiel-Parcours ist in der folgenden Grafik gegeben. Dabei sind freie Zellen weiß gekennzeichnet; solche mit Hindernissen sind schwarz. Darunter ist eine kürzeste Schrittfolge für denselben Parcours dargestellt; die Kreise symbolisieren die Zellen, die vom Roboter betreten werden. Beachten Sie, dass der erste und letzte Schritt jeweils eine Weite von 1 haben und die Weiten aufeinander folgender Schritte sich nur um höchstens 1 unterscheiden.



Das Problem zur Bestimmung einer kürzesten Schrittfolge soll mittels dynamischer Programmierung gelöst werden. Dazu sei $C(i, v)$ die Länge einer kürzesten Schrittfolge von Zelle 0 zur Zelle i, deren letzter Schritt eine Weite von v hat. Der vorletzte Schritt einer zugehörigen Schrittfolge muss demnach die Schrittlänge $v-1$, v oder $v+1$ haben. Dabei ist $v-1$ nur möglich für $v \geq 1$, da negative Schrittweiten nicht erlaubt sind. Außerdem ist $v+1$ nur erlaubt für $v < r$, da der Roboter sonst die maximale Schrittweite überschreiten würde. Der Einfachheit

halber erweitern wir C auf den negativen Bereich durch $C(i, v) = \infty$ für $i < 0$. Damit ergibt sich die folgende rekursive Form, wobei $A(i)$ angibt, ob Zelle i betreten werden kann:

$$C(i,v) = \begin{cases} \infty & \text{falls } A(i) = \text{false}, \\ & \text{oder falls } i > 0 \wedge v = 0, \\ 0 & \text{falls } i = v = 0 \\ \min\{C(i-v, w) \mid \max(0, v-1) \leq w \leq \min(r, v+1)\} + 1 & \text{sonst} \end{cases}$$

Für das obige Beispiel ergibt sich damit für $r = 3$:

C	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$v = 0$	∞	∞	∞	0	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$v = 1$	∞	∞	∞	∞	1	2	3	∞	∞	∞	∞	∞	∞	∞	6	7	7	7
$v = 2$	∞	∞	∞	∞	∞	∞	2	∞	∞	∞	∞	4	∞	5	∞	6	6	7
$v = 3$	∞	∞	∞	∞	∞	∞	∞	∞	∞	3	∞	∞	∞	∞	5	∞	6	6

Die kürzeste Schrittfolge, bei der der Roboter auf Zelle n zum Stehen kommen kann, hat also $C(14, 1) = 7$ Schritte. Der obige Beispiellauf mit sieben Schritten ist damit optimal.

Schreiben Sie eine Klasse `Application` mit einer statischen Methode `generateParcours`, die als Parameter zwei positive Ganzzahlen n und k erwartet und dann zufällig ein `boolean`-Feld der Länge $n+1$ mit genau k Hindernissen erzeugt. Dabei soll `true` bedeuten, dass die entsprechende Zelle betreten werden kann, und `false`, dass auf der entsprechenden Zelle ein Hindernis steht. Beachten Sie, dass weder auf der Startzelle 0 noch auf der Zielzelle n ein Hindernis stehen sollte.

Zur Berechnung einer kürzesten Schrittfolge für ein solches Feld gehen Sie wie folgt vor:

- Schreiben Sie eine `main`-Methode, die die Zahlen n und k und r als Parameter erhält. Erzeugen Sie mit `generateParcours(n, k)` einen entsprechenden Parcours. Lassen Sie sich bis zu einer Feldgröße von 50 das Feld ausgeben, z.B. durch zwei Leerzeichen für freie Zellen, `XX` für Zellen mit Hindernissen und einem senkrechten Strich als Feldgrenzen.

- Schreiben Sie eine Klasse `MinStepCount`, die die weiter oben beschriebene Funktion C repräsentiert. Der Konstruktor der Klasse erhält als Parameter einen Parcours (also ein `boolean`-Feld) sowie die maximale Schrittweite r. Die Funktionswerte von C sollten einmalig berechnet und gespeichert werden, sodass sie danach mit einer Methode `get(i, v)` abgerufen werden können.
  Hinweis: Falls Sie die berechneten Funktionswerte wie oben angedeutet in einem zweidimensionalen Feld speichern, dessen Definitionsbereich nicht genau bei 0 beginnt (i im Beispiel oben), verwenden Sie auch innerhalb der Klasse für jeden Lesezugriff auf das Feld die `get`-Methode und für jeden Schreibzugriff eine entsprechende `set`-Methode. Diese Methoden sollen die nötigen Index-Anpassungen vornehmen, so dass Sie an Stellen, an denen Sie auf $C(i, v)$ zugreifen wollen, auch tatsächlich einfach `get(i, v)` bzw. `set(i, v, value)` aufrufen können. Bei fehlerhafter Index-Berechnung müssen Sie so nachträglich nur Änderungen an zwei Stellen vornehmen.

- Erweitern Sie die Klasse `MinStepCount` um eine Methode `sequenceOfSteps`, die als Parameter eine Zelle x erhält. Diese Methode soll per Backtracking eine optimale Schrittfolge von Zelle 0 nach Zelle x auf Grundlage der C-Werte berechnen. Geben Sie die Schrittfolge als Feld `steps` zurück, wobei `steps`$[i]$ der Position nach dem i-ten Schritt entspricht. Wenn der Roboter aufgrund der Einschränkungen (z.B. mehr Hindernisse direkt hintereinander, als mit einem Schritt übersprungen werden können) das Ziel nicht erreichen kann, soll `null` zurückgegeben werden.

- Erweitern Sie Ihre `main`-Methode um einen Aufruf an `sequenceOfSteps`. Geben Sie die berechnete Schrittfolge in geeigneter Form aus. Mit `<>` für die Positionen, die vom Roboter besucht werden, könnte die Ausgabe für obiges Beispiel wie folgt aussehen:

```
Gegeben: Parcours der Laenge 14 + 1 mit 4 Hindernissen
Die kuerzeste Schrittfolge hat Laenge 7
Parcours: |<><>  <>XXXX<>XX<>XX  <>  <><>|
Schrittfolge: 1 -> 2 -> 3 -> 2 -> 3 -> 2 -> 1
```

Bonusaufgabe 14.2: Hindernislauf in 2D (4 Punkte)

Wir betrachten nun das auf zwei Dimensionen erweiterte Problem aus Aufgabe 1. Auf einem Parcours der Größe $(n+1) \times (m+1)$ sind wieder Hindernisse vorhanden und der Roboter muss nun die Zelle (n, m) von $(0, 0)$ aus erreichen. Der Roboter darf sich in jedem Schritt entweder nach unten oder nach rechts bewegen. Dabei kann er analog zu Aufgabe 1 seine Schrittweite anpassen. In jedem Schritt kann der Roboter aber nur die Schrittweite der Richtung anpassen, in die er sich bewegt. Die aktuelle Schrittweite nach unten ist also unabhängig von der aktuellen Schrittweite nach rechts. Der Roboter wartet am Anfang wieder auf der Startzelle, so dass davon ausgegangen werden kann, dass er auf Zelle $(0, 0)$ mit dem aktuellen Bewegungsvektor $(0, 0)$ startet. Eine Schrittfolge kann also z.B. mit den Schrittweiten $(0, 1) \to (0, 2) \to (0, 3) \to (1, 0)$ beginnen (oder in umgekehrter Reihenfolge enden), aber nicht mit $(0, 1) \to (2, 0) \to (0, 3)$, da der Roboter nicht direkt von Schrittweite 0 auf 2 nach unten beschleunigen kann.

- Überlegen Sie sich zunächst, inwiefern die rekursive Form aus Aufgabe 1 angepasst werden muss.

- Schreiben Sie eine Klasse `Parcours2D`, die das zweidimensionale Problem löst. Es soll also eine gültige Schrittfolge für den Roboter durch einen zweidimensionalen Parcours berechnet werden. Auch hier sollten dynamische Programmierung und Backtracking benutzt werden.

- Ergänzen Sie eine Methode `generateParcours2D`, die einen zweidimensionalen Parcours der Größe $(n + 1) \times (m + 1)$ generiert.

- Erweitern Sie ihre `main`-Methode so, dass nun bei vier `Integer`-Parametern n, m, k und r ein zweidimensionales `boolean`-Feld der Größe $(n + 1) \times (m + 1)$ angelegt und zufällig mit k Hindernissen gespickt wird. Die maximale Schrittweite r gilt nun für beide Richtungen. Lassen Sie schließlich eine optimale Schrittfolge berechnen und ausgeben. Bei einer Feldgröße von maximal 10×10 sollte die Lösung analog zum eindimensionalen Fall visualisiert werden.