Friedrich-Schiller-Universität Jena
Institut für Informatik
Martin Mundhenk
https://www.complexity.uni-jena.de/Python1

# Aufgabenblatt 8, Abgabetermin 6.7.2020

---

Sie können maximal 15 Punkte erreichen.
Bitte beachten Sie die Hinweise zur Abschlussprüfung am Ende des Aufgabenblattes.

---

Auf diesem Aufgabenblatt geht es um „einfache“ Labyrinthe. Das Ziel ist es, in ihnen einen „einfachen“ Weg von oben nach unten zu finden. Auf diesem Weg muss man stets nur nach unten, links oder rechts gehen, aber nie nach oben. Abbildung 1 zeigt ein Beispiel für ein solches Labyrinth. Die Mauern sind schwarz, und es gibt Wege (aus weißen Feldern) von oben nach unten vom Einstieg links oben.
Abbildung 2 zeigt die Struktur eines Labyrinths. Es besteht aus den (roten) Kästchen und einem schwarzen Balken an der linken Seite. In jedem Kästchen ist einer von vier verschiedenen Bausteinen. In der untersten Reihe sehen Sie die verschiedenen Bausteine in den ersten vier Kästchen von links. Wir benennen diese Bausteine durch 0,1,2 und 3 in der Reihenfolge, wie sie dort vorkommen.
Abbildung 3 zeigt das Labyrinth aus Abbildung 2 dargestellt als 2d-Array `grid`. Jede Reihe nebeneinanderliegender Bausteine ist ein Array aus Zahlenwerten 0,1,2,3 für die verschiedenen Bausteine. Die Koordinaten in der bildlichen Darstellung des Bausteins sind seine Indizes in `grid`. Allerdings muss man berücksichtigen, dass die Reihe mit $y$-Koordinate 0 in der bildlichen Darstellung ganz unten ist, während ihr Array aus Bausteinnummern das erste Element von `grid` ist. Der Baustein, der im Bild Koordinaten z.B. (3,1) hat, ist `grid[1][3]`.

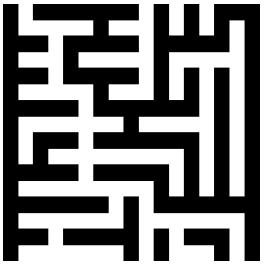


Abb.1
ein Labyrinth

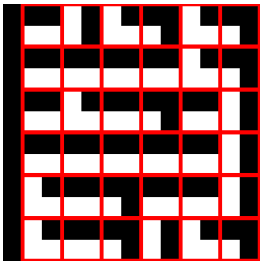


Abb.2
die Struktur

```
[ [0, 1, 2, 3, 0, 2],
  [0, 1, 2, 1, 1, 3],
  [1, 1, 1, 1, 1, 3],
  [1, 0, 1, 2, 1, 3],
  [1, 1, 1, 1, 0, 2],
  [1, 3, 0, 2, 0, 2] ]
```

Abb.3
das 2d-Array aus Arrays
mit Baustein-Nummern

Das Labyrinth in Abbildung 1 besteht aus 8 Reihen mit jeweils 8 Bausteinen. Wir betrachten nur solche quadratischen Labyrinthe. Die Größe eines Labyrinths ist die Anzahl seiner Reihen – das Labyrinth in Abb. 1 hat also Größe 8, und das Labyrinth in Abb. 2 hat Größe 6.

Auf diesem Aufgabenblatt geht es darum,

- ein Modul mit grundlegenden Funktionen zum Malen eines Labyrinths und zum Erzeugen eines zufälligen Labyrinths zu schreiben (zufällige Labyrinthe müssen keinen Weg von oben nach unten enthalten),
- ein Modul mit einer Funktion zu schreiben, mit der man Labyrinthe erzeugt, die tatsächlich einen Weg von oben nach unten enthalten, und
- Funktionen zu schreiben, mit denen man einen Weg durch ein Labyrinth findet.

**Aufgabe 47: Ein Labyrinth zufällig erzeugen und anzeigen** **10 Punkte**

Schreiben Sie ein Modul `laby1.py`, das folgende Funktionen bereitstellt.

| Funktion | Beschreibung |
|---|---|
| `leinwandVorbereiten(n)` | bereitet die Leinwand von `stddraw` zum Malen eines Labyrinths der Größe `n` (`int`) vor |
| `neuesLaby(n,b)` | gibt ein 2d-Array aus `n` Arrays der Länge `n` zurück, bei denen jeder Eintrag `b` ist |
| `zufallsLaby(n)` | gibt ein 2d-Array aus `n` Arrays der Länge `n` zurück, bei denen jeder Eintrag zufällig aus 0,1,2,3 gewählt wurde |
| `maleLaby(l)` | malt Labyrinth `l` (2d-Array aus `int`) auf die leere Leinwand von `stddraw` |

Schreiben Sie eine Testfunktion für das Modul, das eine Zahl `n` einliest, ein Labyrinth der Größe `n` aus Bausteinen mit Nummer 2 anzeigt und anschließend ein Zufalls-Labyrinth der Größe `n` anzeigt.

**Aufgabe 48: Ein Labyrinth erzeugen, durch das es einen Weg gibt** **5 Punkte**

Wir wollen nun Labyrinthe erzeugen, durch die es einen einfachen Weg gibt. Dazu beginnen wir mit einem Labyrinth, das nur aus Baustein 2 besteht. Abb.5 zeigt zwei Reihen eines solchen Labyrinths der Größe 10. Die obere Hälfte jeder Reihe ist durchgehend schwarz, und die untere Hälfte ist schwarz-weiß gemustert. In jeder Reihe geht man wie folgt vor. Man wählt zufällig eine Zahl und „fräst“ in der gemusterten Hälfte von links nach rechts diese Anzahl von schwarzen Quadraten weg (d.h. man ersetzt Baustein 2 durch Baustein 1). Die Ränder müssen natürlich stehen bleiben. Abb.6 zeigt, was aus Abb.5 entsteht, wenn 2 schwarze Quadrate weggefräst wurden. Der bearbeitete Bereich ist rot markiert.
Abschließend muss im bearbeiteten Bereich noch eine Öffnung nach oben gemacht werden. Die Stelle dazu wird zufällig gewählt. Das Ergebnis sieht man in Abb.7.

Abb.5

Abb.6

Abb.7

Das schwarze Quadrat am Ende des bearbeiteten Bereichs lässt man stehen und setzt die Arbeit dahinter fort. Man wählt also wieder zufällig einen Bereich, in dem die schwarzen Quadrate weggefräst werden und eine Öffnung nach oben gemacht wird.

Abb.8

Abb.9

Abb.10

Abb.8–10 zeigen, was dabei entstehen kann. Wenn man mit der Reihe fertig ist, setzt man die Arbeit in der nächsten Reihe fort.
Schreiben Sie ein Modul `laby2.py`, das folgende Funktion bereitstellt.

| Funktion | Beschreibung |
|---|---|
| `erzeugeLaby(n)` | gibt ein 2d-Array für ein Labyrinth der Größe `n` (`int`) zurück |

Schreiben Sie eine Testfunktion für das Modul, die eine Zahl `n` einliest, ein Laybrinth der Größe `n` erzeugt und es anzeigt. Sie können das Modul `laby1.py` aus der vorigen Aufgabe benutzen.

**Aufgabe 49: Ein Labyrinth wässern** **5 Punkte**

Wir wollen nun feststellen, ob es durch ein in den vorigen Aufgaben erzeugtes Labyrinth einen Weg von oben nach unten gibt. Das kann man nicht immer mit dem bloßen Auge erkennen – also muss ein Programm die Arbeit machen. Dafür kann man folgende Idee programmieren: man schüttet oben Wasser in das Labyrinth und schaut nach, ob das Wasser unten wieder herauskommt. Von einem „nassen“ Baustein kann Wasser nach unten fließen, wenn darunter ein Baustein liegt, der oben eine Öffnung habt (Baustein 0 und Baustein 3). Von einem „nassen“ Baustein kann Wasser nach links fließen, wenn dort Baustein 0 oder 1 liegt. Von einem „nassen“ Baustein 0 oder 1 kann das Wasser auch nach rechts fließen.

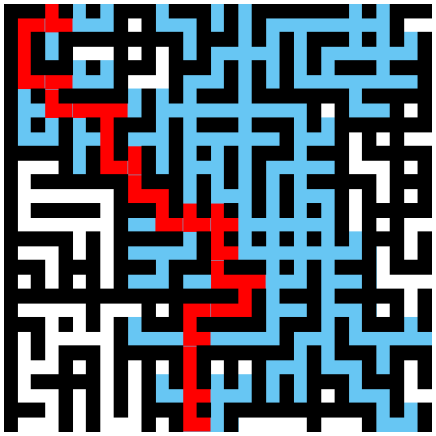
Für jeden Baustein $x$ muss man sich beim Wässern den Baustein merken, von dem aus (erstmals) Wasser zu ihm geflossen ist – diesen Baustein nennen wir den *Vorgänger von* $x$. Nun kann man einen nassen Baustein in der untersten Reihe suchen und von ihm aus immer zu den Vorgängern zu gehen, bis man nicht mehr weiterkommt. Diese Steine bilden dann einen Weg durch das Labyrinth.

Schreiben Sie ein Modul `laby3.py`, das folgende Funktion bereitstellt.

| Funktion | Beschreibung |
|---|---|
| `weg(l)` | sucht und malt einen Weg durch das Labyrinth `l` |

Schreiben Sie eine Testfunktion für das Modul, die eine Zahl `n` einliest, ein Laybrinth oder ein Zufalls-Labyrinth der Größe `n` erzeugt und anzeigt, und anschließend einen Weg durch das Labyrinth anzeigt.

In der folgenden Abbildung sind die beim Wässern nass gewordenen Steine (blau) und ein dadurch bestimmter Weg (rot) durch das Labyrinth dargestellt.



---

## Die Abschlussprüfung

Die Abschlussprüfung dieses Moduls besteht traditionell darin, dass die Teilnehmenden (1) eine Programmieraufgabe lösen und (2) ihre Module in einer mündlichen Prüfung vorstellen. Die Programmieraufgaben stammen entweder aus einem Aufgabenkatalog oder sind selbst gestellte Aufgaben (in Abstimmung mit mir).

Dieses Semester werden wir auf den mündlichen Teil (2) verzichten. Die Abschlussprüfung besteht dann darin, dass Sie eine Programmieraufgabe lösen und die Module mit einer kurzen schriftlichen Erläuterung (grobe Programmstruktur, API, ggf. grobe Struktur der Funktionen, kommentierte Programme) abgeben. Der Abgabetermin ist der 15. September 2020. Wir werden Ihre Lösungen benoten und Ihnen die Noten bekanntgeben. Der Aufgabenkatalog wird in der vorletzten Vorlesungswoche bereitgestellt.

Wenn Sie einen späteren Abgabetermin brauchen, dann melden Sie sich bitte bis zum Ende der Vorlesungszeit bei mir.