Aufgabe 4: Nandu

Teilnahme-ID: 69408

Bearbeiter dieser Aufgabe:
Tim Krome

31. Oktober 2023

Lösungsidee

Modellierung

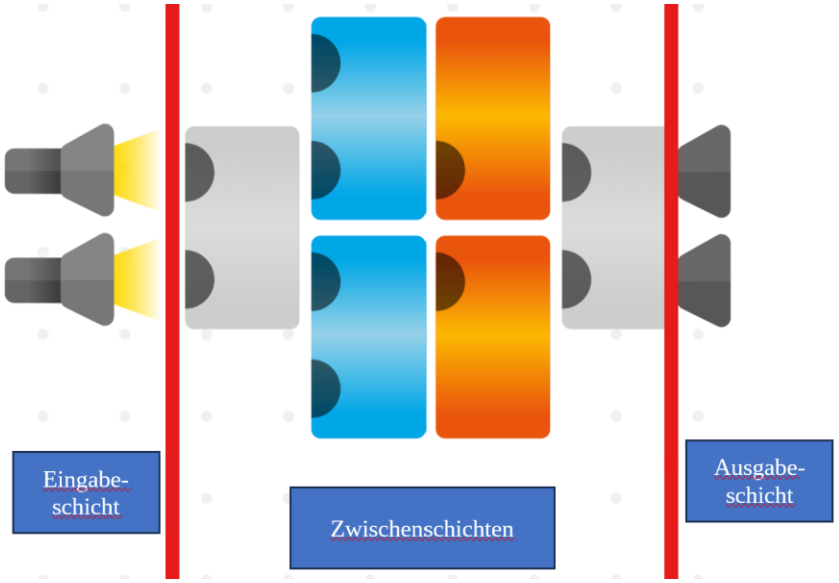
Eine Nandu-Konstruktion besteht aus Bausteinen dreierlei Arten (im Folgenden auch als „Komponenten“ bezeichnet), die zu einer Konstruktion zusammengesteckt werden können. Jede Komponente hat zwei Sensoren, über die sie zwei Lichtsignale empfangen kann. Sie hat außerdem zwei LEDs, mit denen sie abhängig von den empfangen Lichtsignalen zwei eigene Lichtsignale weitergibt. Teil der Konstruktion sind auch die ein- und ausschaltbaren Taschenlampen, mit denen die vorderen Komponenten der Konstruktion angestrahlt werden können. Außerdem ist genau festgelegt, bei welchen LEDs der hinteren Komponenten der Leucht-Status betrachtet (also, ob die LED an oder aus ist) betrachtet werden soll und bei welchen nicht.

Eine Nandu-Konstruktion lässt sich dabei schichtweise darstellen und speichern. Allgemein lässt sich sagen, dass eine vollständige Konstruktion immer so aufgebaut ist:

- Ganz links ist eine Schicht, in der sich die Ein- und Ausschaltbaren Taschenlampen befinden. Diese Schicht wird im Folgenden als Eingabe-Schicht bzw. Input-Schicht bezeichnet, da in ihr die Lichtsignale von den Taschenlampen in die Konstruktion eingeführt werden.

- Ganz rechts ist eine Schicht, in der sich die Ausgabe-LEDs befinden. Genau genommen ist in dieser Schicht festgelegt, bei welchen LEDs der letzten Schicht der Output bzw. das ausgehende Lichtsignal betrachtet werden soll. Diese Schicht wird daher im Folgenden als Ausgabe-Schicht bzw. Output-Schicht bezeichnet.
- Zwischen Eingabe- und Ausgabeschicht befinden sich n Schichten, in denen die einzelnen Komponenten zu einer Konstruktion zusammengesteckt ist. Diese Schichten werden im Folgenden als Zwischenschichten bezeichnet. Das einer Zwischenschicht empfangene Signal wird über die Komponenten der Schicht an die jeweils nächste Zwischenschicht weitergegeben.

In dieser Grafik wird die Einteilung in Eingabeschicht, Ausgabeschicht und Zwischenschichten verdeutlicht:



Bestimmung der Signale der Ausgabe-Schicht

Es soll bestimmt werden, wie die LEDs der Ausgabe-Schicht reagieren, wenn die Taschenlampen der Eingabe-Schicht ein- und ausgeschaltet werden. In einer Tabelle soll anschließend dokumentiert werden, wie die LEDs reagieren, wenn die verschiedenen Taschenlampen an- bzw. ausgeschaltet sind. Hierfür müssen zunächst alle möglichen Kombinationen aus ein- und ausgeschalteten Taschenlampen (im Folgenden als TL abgekürzt) bestimmt werden. Anders gesagt: Es müssen alle Möglichkeiten bestimmt werden, den einzelnen TL einen Status (ein oder aus) zuzuweisen. Gibt es nur zwei TL, dann existieren hierfür nur vier Möglichkeiten:

Möglichkeit 1: TL 1 und 2 sind aus
Möglichkeit 2: TL 1 ist an, TL 2 ist aus
Möglichkeit 3: TL 1 ist aus, TL 2 ist an
Möglichkeit 4: TL 1 und 2 sind an

Bei mehr als zwei Taschenlampen gibt es deutlich mehr Möglichkeiten. Für *n* Taschenlampen können alle existierenden Möglichkeiten bestimmt werden, indem zuerst für TL 1 beide Möglichkeiten (an / aus) notiert werden, dann zu jeder Möglichkeit von TL 1 die beiden

Möglichkeiten für TL 2 notiert werden, dann zu jeder Möglichkeit von TL 2 die beiden Möglichkeiten für TL 3 notiert werden usw., bis man bei TL *n* angekommen ist.

Anschließend muss für jede so bestimmte Taschenlampenzustandskombination ermittelt werden, wie die LEDs der Ausgabeschicht auf die Taschenlampen-Signale reagieren werden.

Um dies zu bestimmen muss simuliert werden, wie sich das Signal aus der Eingabeschicht durch die Konstruktion „bewegt“. Das heißt: Zuerst wird simuliert, wie die einzelnen Komponenten der Zwischenschicht 1 die Signale der Eingabeschicht verarbeiten. Die von den einzelnen Komponenten ausgegebenen Lichtsignale (bzw. die Ausgabe der Zwischenschicht 1) werden anschließend an Zwischenschicht 2 weitergegeben, die diese Signale als Eingabesignale empfängt. Dabei muss darauf geachtet werden, dass die Ausgabesignale von Zwischenschicht 1 an die richtigen Komponenten bzw. Sensoren von Zwischenschicht 2 weitergegeben werden[1]. Zwischenschicht 2 verarbeitet die empfangenen Signale und gibt ihre Ausgabe an Schicht 3 weiter, die sie wiederum als Signale verarbeitet und ihre Ausgabe an Schicht 4 weitergibt usw. So kann die von den Taschenlampen in die Konstruktion eingeführte Eingabe durch die Schichten der Konstruktion propagiert werden, bis die Output-Schicht erreicht ist. Die Ausgabe der Output-Schicht wird notiert bzw. die Zustände der LEDs werden gespeichert.

Auf die Art kann eine Tabelle mit allen möglichen Taschenlampenzustandskombinationen und den zugehörigen LED-Ausgaben erstellt werden.

Umsetzung

Die Lösungsidee wird in Python 3.10 implementiert. Zum Darstellen eines Lichtsignals werden in der Implementierung grundsätzlich booleans verwendet. True bedeutet dabei, dass ein Lichtsignal vorhanden ist bzw. dass die LED an ist. False bedeutet, dass kein Lichtsignal vorhanden ist bzw. die LED aus ist.

Einlesen der Eingabedatei

Der Dateipfad der .txt-Eingabedatei wird vom Nutzer als erstes Kommandozeilenargument angegeben. Das Programm liest die Textdatei als String ein und splittet sie in eine Liste namens *input_lines* auf, die die Zeilen der Textdatei enthält. Die Breite *n* und Höhe *m* der Konstruktion werden in entsprechend benannten Variablen *m* und *n* gespeichert.

```
import sys

# Textdatei einlesen
# sys.argv[1] ist das erste Kommandozeilenargument
with open(sys.argv[1]) as f:
    input_lines = f.read().split("\n")

(...)

line = input_lines.pop(0).split(" ")
```

[1] Beim richtigen „Sensor“ handelt es sich um den Sensor, der rechts von der LED ist, die das Signal ausgibt. Dieser Sensor empfängt nämlich das Signal der LED.

```
n = int(line[0])
m = int(line[1])
```

Definieren einer Klasse zum Modellieren einer Schicht

Es wird eine Klasse namens *Layer* erstellt, die eine Schicht einer Konstruktion darstellt. Die Klasse ist außerdem mit Funktionen ausgestattet, die zum Verarbeiten von Signalen dienen.

Die Klasse hat ein *Layer.aufbau* Attribut, in dem der Aufbau der Schicht gespeichert ist. In der Liste sind dabei die Komponenten, aus denen die Schicht besteht, in der richtigen Reihenfolge als Strings in Form von folgenden **Kürzeln** gespeichert:

- „WW“: Dieser Listeneintrag bedeutet, dass sich an dieser Stelle ein weißer Baustein befindet. Weiße Bausteine haben eine Breite von 2 (die Breite wird an der Breite der Taschenlampen gemessen, d.h. eine Breite von 1 bedeutet, dass der Baustein so breit ist wie eine Taschenlampe).
- „BB“: Dieser Eintrag bedeutet, dass sich an der Stelle ein blauer Baustein befindet. Blaue Bausteine haben eine Breite von 2.
- „X“: Dieser Eintrag bedeutet, dass sich an der Stelle kein Baustein befindet. Der Bereich, in dem sich kein Baustein befindet, hat dabei eine Breite von 1.
- „Q*n*“: Dieser Eintrag bedeutet, dass sich an der Stelle die n-te Taschenlampe.
- „L*n*“: Dieser Eintrag bedeutet, dass sich an der Stelle die n-te LED befindet. LEDs haben eine Breite von 1.
- „Rr“: Dieser Eintrag bedeutet, dass sich an der Stelle ein roter Baustein befindet, dessen Sensor unten ist.
- „rR“: Dieser Eintrag bedeuetet, dass sich an der Stelle ein roter Baustein befindet, dessen Sensor oben ist.

Auf die Art kann der Aufbau einer Schicht gespeichert werden, ohne dass für die einzelnen Bausteinarten Klassen definiert werden müssen.

Die Klasse hat außerdem diese Funktionen:

- *Layer.X(inputs, outputs)* und *Layer.WW(inputs, outputs)* und *Layer.BB(inputs, outputs)* und *Layer.Rr(inputs, outputs)* und *Layer.rR(inputs, outputs)*: Jede Funktion repräsentiert eine Bausteinart. Der Name der Funktion entspricht dabei dem Kürzel der entsprechenden Bausteinart (siehe oben).
  Diese Funktionen funktionieren alle nach demselben Prinzip: Jede der Funktionen entnimmt und entfernt die ersten beiden Elemente aus *inputs* und verarbeitet sie so als Signale, wie die repräsentierte Bausteinart sie verarbeiten würde. Die Signale, die der repräsentierte Baustein ausgeben würde, werden an *outputs* angehängt, anschließend werden die veränderten Listen *inputs* und *outputs* zurückgegeben.
  Beispielsweise nimmt die Funktion *Layer.BB(inputs, outputs)* die ersten beiden Elemente aus *inputs* und verarbeitet sie so als Signale, wie es ein blauer Lichtbaustein tut: Sie gibt die empfangenen Signale, ohne sie zu verändern, weiter:

```
def BB(self, inputs, outputs):
    sensor1, sensor2 = inputs[:2]
    outputs = outputs + [sensor1, sensor2]
    return inputs[2:], outputs
```

Die *Layer.Rr* und *Layer.rR* Funktionen hingeben entnehmen ebenfalls zwei Elemente aus *inputs*, verarbeiten aber nur eines der beiden Signale, da rote Bausteine nur einen Sensor haben. *Layer.X(inputs, outputs)* verarbeitet Signale, die auf Plätze treffen, an denen sich kein Baustein befindet: Sie annulliert ganz einfach das empfangene Signal und fügt als Ausgabe *False* zu *outputs* hinzu.
Diese Funktionen ermöglichen es der Schicht, basierend auf Eingabesignalen die Ausgabesignale der Schicht zu bestimmen.

- *Layer.forward(inputs):* Diese Funktion kann verwendet werden, wenn es sich bei der durch das Layer-Objekt repräsentierten Schicht um eine Zwischenschicht handelt (siehe Kapitel Lösungsidee). Die Funktion nimmt die Inputs, die die repräsentierte Schicht empfängt, und bestimmt die Ausgabe der Schicht. Hierfür initialisiert sie zuerst eine leere Liste namens *outputs*, in der die von der Schicht ausgegebenen Signale gespeichert werden sollen. Anschließend iteriert sie über die Komponenten, aus denen die Schicht besteht, und wendet für jede Komponente die entsprechende Funktion (*Layer.BB*, *Layer.WW*, *Layer.Rr* oder *Layer.rR*) auf *inputs* und *outputs* an:

```
def forward(self, inputs):
     outputs = []
     for component in self.aufbau:
         if component == "X":
             inputs, outputs = self.X(inputs, outputs)
         if component == "WW":
             inputs, outputs = self.WW(inputs, outputs)
         if component == "BB":
             inputs, outputs = self.BB(inputs, outputs)
         if component == "Rr":
             inputs, outputs = self.Rr(inputs, outputs)
         if component == "rR":
             inputs, outputs = self.rR(inputs, outputs)
     return outputs
```

  Bei jeder Iteration der for-Schleife wird die Funktion, die die Funktionsweise der jeweils betrachteten Komponente darstellt, auf *inputs* und *outputs* angewandt. Auf die Art wird *inputs* bei jeder Iteration kleiner und *outputs* entsprechend größer (die angewendeten Funktionen entnehmen die ersten zwei Elemente aus *inputs* and fügen die Ausgabesignale der jeweiligen Komponente zu *outputs* hinzu). So wird die Ausgabe der Schicht schrittweise bestimmt. Die Funktion gibt *outputs* zurück.
- *Layer.input_layer(inputs)* funktioniert ähnlich wie Layer.forward(inputs), allerdings ist sie dazu da, die Inputs der Input-Schicht, also die Zustände der Taschenlampen zu verarbeiten und die Outputs der Input-Schicht zu bestimmen bzw. die Zustände der Taschenlampen in ein Format zu bringen, dass die nächste Schicht (also die erste Zwischensicht) einlesen kann. Diese Funktion darf nur angewendet werden, wenn die vom Layer-Objekt repräsentierte Schicht die Input-Schicht ist. Die Funktion nimmt als Argument die Liste *inputs*, in der sich die Zustände der Taschenlampen befinden. Das n-te Element von *inputs* beschreibt den Zustand der n-te Taschenlampe als Boolean (an = True, aus = False). Die Funktion iteriert über alle Komponenten der Input-Schicht; wenn sie auf eine Taschenlampe stößt greift sie auf den Zustand der Taschenlampe zu und hängt ihn an *outputs* an, ansonsten hängt sie False an *outputs* an (wo keine Taschenlampe ist, da kann auch kein Signal eingehen):

```
    def input_layer(self, inputs):
        outputs = []
        for component in self.aufbau:
            if component == "X":
                outputs.append(False)
            if "Q" in component:
                q_index = int(component[1:])-1
                outputs.append(inputs[q_index])
        return outputs
```

- *Layer.output_layer(inputs)* darf nur angewendet werden, wenn die repräsentierte Schicht die Ausgabeschicht ist. Sie nimmt die Ausgaben der letzten Zwischenschicht als Inputs und ordnet sie den entsprechenden LEDs zu. Zurückgegeben wird eine Liste, in der die Zustände der Output-LEDs angegeben sind. Element *n* in der zurückgegebenen Liste beschreibt dabei den Zustand der *n*-ten LED:

```
    def output_layer(self, inputs):
        outputs = []
        for component in self.aufbau:
            if "L" in component:
                l_index = int(component[1:])-1
                while len(outputs) < l_index + 1:
                    outputs.append(None)
                outputs[l_index] = inputs.pop(0)
            else:
                inputs.pop(0)
        return outputs
```

Einlesen und Speichern einer Konstruktion

Die Konstruktion wird eingelesen und in einer Liste namens *konstruktion* gespeichert. Die Liste enthält dabei die Schichten der Konstruktion, jede Schicht wird durch ein *Layer*-Objekt repräsentiert. Das erste Element aus *konstruktion* entspricht dabei der Eingabschicht, das letzte Element der Ausgabeschicht. Beim Einlesen der Konstruktion wird der Inhalt jeder Schicht in das Format gebracht, in dem es anschließend im Layer-Objekt unter dem *Layer.aufbau* Attribut gespeichert wird.
Parallel dazu wird auch die Anzahl an LEDs und Taschenlampen ermittelt.

```
konstruktion = []
num_taschenlampen = 0
num_leds = 0
for i in range(m):
    line = input_lines.pop(0).replace("  ", " ").split(" ")[:n]
    aufbau = []
    while line != []:
        component = line.pop(0)
        if "Q" in component:
            num_taschenlampen += 1
        elif "L" in component:
            num_leds += 1
        elif not component == "X":
```

```
            component += line.pop(0)
        aufbau.append(component)
    konstruktion.append(Layer(aufbau))
```

Zustände der LEDs für jede mögliche Taschenlampenkombination ermitteln und ausgeben

Zunächst werden alle möglichen Kombinationen der Taschenlampenzustände ermittelt. Diese werden in der verschachtelten Liste *kombinationen* gespeichert, jedes Element aus *kombinationen* stellt eine mögliche Zustands-Kombination der Taschenlampen dar. Zum Finden aller möglichen Kombinationen wird das in der Lösungsidee vorgestellte Verfahren mithilfe von zwei verschachtelten for-Schleifen umgesetzt. In jeder Iteration der äußeren for-Schleife werden zu jedem Element aus *kombinationen* die Möglichkeiten für den Zustand der jeweiligen Taschenlampe (True = an, False = aus) hinzugefügt:

```
# Alle möglichen Kombinationen der Taschenlampen (ein / aus) ermitteln:
kombinationen = [[]]
for t in range(num_taschenlampen):
    for k in list(kombinationen):
        kombinationen.remove(k)
        kombinationen.append(k+[False])
        kombinationen.append(k+[True])
```

Anschließend wird für jede mögliche Kombination der Taschenlampen die Zustände der LEDs ermittelt, indem zunächst der Input von der Input-Schicht mit *konstruktion[0].input_layer(kombination)* verarbeitet wird, danach der Output der Input Schicht mit folgendem Code in der Konstruktion vorwärts propagiert wird …

for layer in konstruktion[1:-1]:
  output = layer.forward(output)

… bis der Output der letzten Zwischenschicht schließlich von der Output-Schicht mit *konstruktion[-1].output.layer(output)* verarbeitet wird. Die Output-Schicht gibt die Zustände der LEDs zurück, die in der Liste *leds* gespeichert wird. Am Ende enthält des n-te Element vom *leds* die LED-Zustände für die n-te Taschenlampenkombination aus *kombinationen*.

```
leds = []
for k in kombinationen:
    output = konstruktion[0].input_layer(k)
    for layer in konstruktion[1:-1]:
        output = layer.forward(output)
    output = konstruktion[-1].output_layer(output)
    leds.append(output)
```

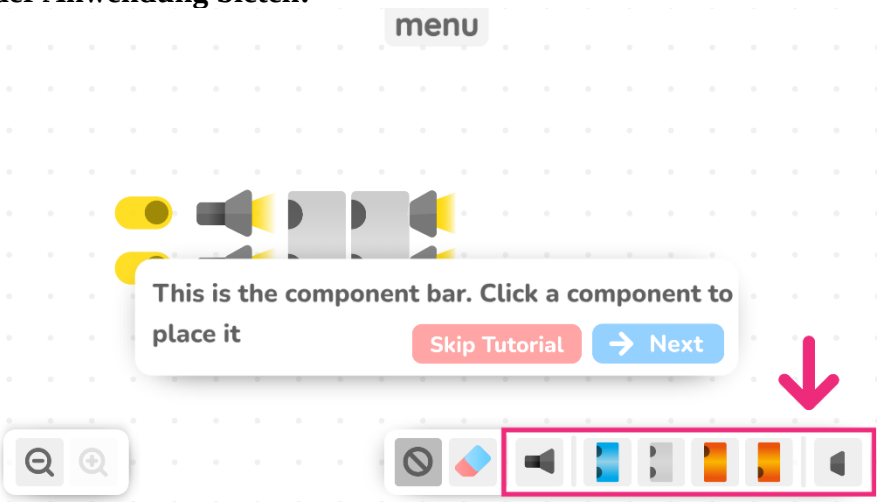
Die ermittelten Zustände werden anschließend in Tabellenform ausgegeben. In jeder Zeile der Tabelle sind die Zustände der Taschenlampen und LEDs festgehalten.

Zusatzaufgabe

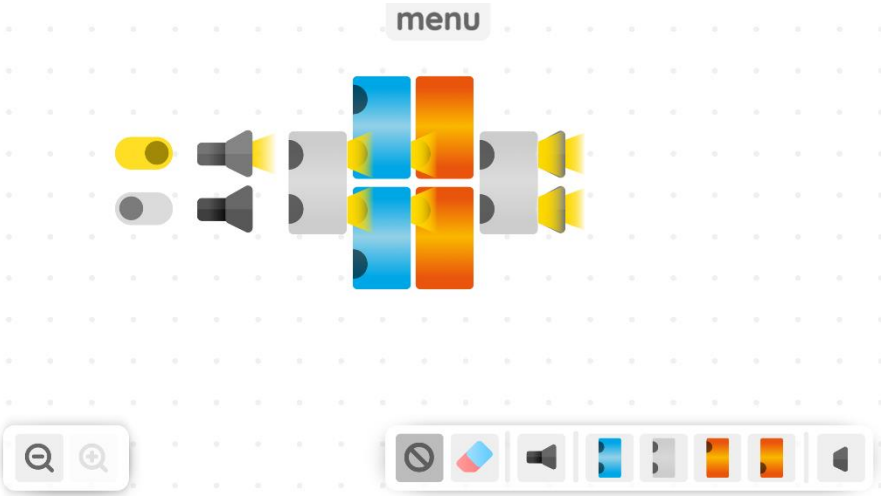
Ich habe für die Zusatzaufgabe eine Browseranwendung erstellt, mit der Nandu-Konstruktionen erstellt werden können und die Funktionsweise der erstellten Konstruktion angezeigt wird. Die Browseranwendung kann aufgerufen werden, indem die index.html Datei aus dem Ordner *A4_Nandu_Zusatzaufgabe* im Browser geöffnet wird. Die index.html Datei ist allein lauffähig.

Getestete Browser: Google Chrome, Microsoft Edge, Opera. Sehr wahrscheinlich funktioniert die Anwendung in allen modernen Browsern
Technische Daten: Die Anwendung wurde mit Scratch erstellt, dabei wurde der Scratch-Mod TurboWarp (turbowarp.org) verwendet, damit die Bühnengröße angepasst werden kann. Das Scratch Projekt wurde anschließend mit dem TurboWarp Packager (packager.turbowarp.org) nach HTML konvertiert. Das ursprüngliche Scratch-Projekt ist in einem Unterordner von *A4_Nandu_Zusatzaufgabe* beigelegt. Es ist kein asset / keine dependency der lauffähigen HTML Datei! Die index.html ist allein lauffähig.
Der gesamte Programmcode sowie alle Bilder wurden von mir erstellt.

Screenshots von der erstellten Browseranwendung, die einen Einblick in die Grundfunktionen der Anwendung bieten:



Nach Öffnen der App gibt ein Tutorial dem Nutzer einen Überblick über die Grundfunktionen



Eine in der Anwendung zusammengestellte Nandu-Konstruktion

Die Menüleiste stellt weitere Funktionen zur Verfügung, so kann beispielweise ein Joystick aktiviert werden. Außerdem kann aktiviert werden, dass jede Schicht ihren Output direkt in der Konstruktion anzeigt. Ferner können bestehende Konstruktionen als Strukturen oder über einen Save Code importiert / exportiert werden.

Beispiele

Die in Beispiel 1 - 5 verwendeten Eingabedateien stammen von der BWinf-Webseite. Zu Beispiel *x* gehört stets die Eingabedatei nandu*x*.txt.

Beispiel 1

Eingabedatei:
nandu1.txt

Inhalt der Eingabedatei:
```
4 6
X  Q1 Q2 X
X  W  W  X
r  R  R  r
X  B  B  X
X  W  W  X
X  L1 L2 X
```

Ausgabe:
```
Q1   | Q2   | L1   | L2
Aus  | Aus  | An   | An
Aus  | An   | An   | An
An   | Aus  | An   | An
An   | An   | Aus  | Aus
```

Beispiel 2

Eingabedatei:
nandu2.txt

Inhalt der Eingabedatei:
```
6 8
X  X  Q1 Q2 X  X
X  X  B  B  X  X
X  X  W  W  X  X
X  r  R  R  r  X
r  R  W  W  R  r
X  W  W  B  B  X
X  X  B  B  X  X
X  X  L1 L2 X  X
```

Ausgabe:

```
Q1    | Q2    | L1    | L2
Aus   | Aus   | Aus   | An
Aus   | An    | Aus   | An
An    | Aus   | Aus   | An
An    | An    | An    | Aus
```

Beispiel 3

Eingabedatei:
nandu3.txt

Inhalt der Eingabedatei:

```
6 8
X  X  Q1 Q2 X  X
X  X  B  B  X  X
X  X  W  W  X  X
X  r  R  R  r  X
r  R  W  W  R  r
X  W  W  B  B  X
X  X  B  B  X  X
X  X  L1 L2 X  X
```

Ausgabe:

```
Q1    | Q2    | Q3    | L1    | L2    | L3    | L4
Aus   | Aus   | Aus   | An    | Aus   | Aus   | An
Aus   | Aus   | An    | An    | Aus   | Aus   | Aus
Aus   | An    | Aus   | An    | Aus   | An    | An
Aus   | An    | An    | An    | Aus   | An    | Aus
An    | Aus   | Aus   | Aus   | An    | Aus   | An
An    | Aus   | An    | Aus   | An    | Aus   | Aus
An    | An    | Aus   | Aus   | An    | An    | An
An    | An    | An    | Aus   | An    | An    | Aus
```

Beispiel 4

Eingabedatei:
nandu4.txt

Inhalt der Eingabedatei:

```
8 8
X  X  Q1 Q2 Q3 Q4 X  X
X  r  R  B  B  R  r  X
X  X  W  W  W  W  X  X
X  r  R  B  B  R  r  X
r  R  W  W  B  B  R  r
X  W  W  W  W  W  W  X
X  X  B  B  X  X  X  X
X  X  L1 L2 X  X  X  X
```

Ausgabe:

Q1	Q2	Q3	Q4	L1	L2
Aus	Aus	Aus	Aus	Aus	Aus
Aus	Aus	Aus	An	Aus	Aus
Aus	Aus	An	Aus	Aus	An
Aus	Aus	An	An	Aus	Aus
Aus	An	Aus	Aus	An	Aus
Aus	An	Aus	An	An	Aus
Aus	An	An	Aus	An	An
Aus	An	An	An	An	Aus
An	Aus	Aus	Aus	Aus	Aus
An	Aus	Aus	An	Aus	Aus
An	Aus	An	Aus	Aus	An
An	Aus	An	An	Aus	Aus
An	An	Aus	Aus	Aus	Aus
An	An	Aus	An	Aus	Aus
An	An	An	Aus	Aus	An
An	An	An	An	Aus	Aus

Beispiel 5

Eingabedatei:
nandu5.txt

Inhalt der Eingabedatei:

```
22 14
X  X  X  X  X  Q1 Q2 X  X  Q3 Q4 X  X  Q5 Q6 X  X  X  X  X  X  X
X  X  X  X  X  R  r  X  X  r  R  X  X  R  r  X  X  X  X  X  X  X
X  X  X  X  r  R  R  r  r  R  R  r  r  R  R  r  X  X  X  X  X  X
X  X  X  X  W  W  B  B  W  W  B  B  W  W  B  B  X  X  X  X  X  X
X  X  X  r  R  B  B  B  B  B  B  B  B  B  B  R  r  X  X  X  X  X
X  X  r  R  B  B  X  B  B  B  B  B  B  B  B  X  R  r  X  X  X  X
X  r  R  B  B  R  r  B  B  B  B  B  B  B  B  r  R  R  r  X  X  X
X  X  B  B  W  W  B  B  X  B  B  B  B  B  B  W  W  B  B  X  X  X
X  r  R  W  W  W  W  R  r  W  W  W  W  W  W  W  W  W  W  X  X  X
X  W  W  B  B  W  W  X  B  B  W  W  B  B  W  W  B  B  R  r  X  X
r  R  B  B  B  B  R  r  X  B  B  B  B  B  B  B  B  B  B  R  r  X
X  B  B  B  B  B  B  R  r  B  B  B  B  B  B  B  B  B  B  W  W  X
r  R  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  B  R  r
L1 X  X  X  X L2  X  X  X  X L3  X  L4 X  X  X  X  X  X  X  X  L5
```

Ausgabe:

Q1	Q2	Q3	Q4	Q5	Q6	L1	L2	L3	L4	L5
Aus	Aus	Aus	Aus	Aus	Aus	Aus	Aus	Aus	An	Aus
Aus	Aus	Aus	Aus	Aus	An	Aus	Aus	Aus	An	Aus
Aus	Aus	Aus	Aus	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	Aus	Aus	An	An	Aus	Aus	Aus	An	An
Aus	Aus	Aus	An	Aus	Aus	Aus	Aus	An	Aus	Aus
Aus	Aus	Aus	An	Aus	An	Aus	Aus	An	Aus	Aus
Aus	Aus	Aus	An	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	Aus	An	An	An	Aus	Aus	Aus	An	An
Aus	Aus	An	Aus	Aus	Aus	Aus	Aus	Aus	An	Aus
Aus	Aus	An	Aus	Aus	An	Aus	Aus	Aus	An	Aus
Aus	Aus	An	Aus	An	Aus	Aus	Aus	Aus	An	An
Aus	Aus	An	Aus	An	An	Aus	Aus	Aus	An	An
Aus	Aus	An	An	Aus	Aus	Aus	Aus	An	Aus	Aus
Aus	Aus	An	An	Aus	An	Aus	Aus	An	Aus	Aus

```
Aus | Aus | An  | An  | An  | Aus | Aus | Aus | Aus | An  | An
Aus | Aus | An  | An  | An  | An  | Aus | Aus | Aus | An  | An
Aus | An  | Aus | Aus | Aus | Aus | Aus | Aus | Aus | An  | Aus
Aus | An  | Aus | Aus | Aus | An  | Aus | Aus | Aus | An  | Aus
Aus | An  | Aus | Aus | An  | Aus | Aus | Aus | Aus | An  | An
Aus | An  | Aus | Aus | An  | An  | Aus | Aus | Aus | An  | An
Aus | An  | Aus | An  | Aus | Aus | Aus | Aus | An  | Aus | Aus
Aus | An  | Aus | An  | Aus | An  | Aus | Aus | An  | Aus | Aus
Aus | An  | Aus | An  | An  | Aus | Aus | Aus | Aus | An  | An
Aus | An  | Aus | An  | An  | An  | Aus | Aus | Aus | An  | An
Aus | An  | An  | Aus | Aus | Aus | Aus | Aus | Aus | An  | Aus
Aus | An  | An  | Aus | Aus | An  | Aus | Aus | Aus | An  | Aus
Aus | An  | An  | Aus | An  | Aus | Aus | Aus | Aus | An  | An
Aus | An  | An  | Aus | An  | An  | Aus | Aus | Aus | An  | An
Aus | An  | An  | An  | Aus | Aus | Aus | Aus | An  | Aus | Aus
Aus | An  | An  | An  | Aus | An  | Aus | Aus | An  | Aus | Aus
Aus | An  | An  | An  | An  | Aus | Aus | Aus | Aus | An  | An
Aus | An  | An  | An  | An  | An  | Aus | Aus | Aus | An  | An
An  | Aus | Aus | Aus | Aus | Aus | An  | Aus | Aus | An  | Aus
An  | Aus | Aus | Aus | Aus | An  | An  | Aus | Aus | An  | Aus
An  | Aus | Aus | Aus | An  | Aus | An  | Aus | Aus | An  | An
An  | Aus | Aus | Aus | An  | An  | An  | Aus | Aus | An  | An
An  | Aus | Aus | An  | Aus | Aus | An  | Aus | An  | Aus | Aus
An  | Aus | Aus | An  | Aus | An  | An  | Aus | An  | Aus | Aus
An  | Aus | Aus | An  | An  | Aus | An  | Aus | Aus | An  | An
An  | Aus | Aus | An  | An  | An  | An  | Aus | Aus | An  | An
An  | Aus | An  | Aus | Aus | Aus | An  | Aus | Aus | An  | Aus
An  | Aus | An  | Aus | Aus | An  | An  | Aus | Aus | An  | Aus
An  | Aus | An  | Aus | An  | Aus | An  | Aus | Aus | An  | An
An  | Aus | An  | Aus | An  | An  | An  | Aus | Aus | An  | An
An  | Aus | An  | An  | Aus | Aus | An  | Aus | An  | Aus | Aus
An  | Aus | An  | An  | Aus | An  | An  | Aus | An  | Aus | Aus
An  | Aus | An  | An  | An  | Aus | An  | Aus | Aus | An  | An
An  | Aus | An  | An  | An  | An  | An  | Aus | Aus | An  | An
An  | An  | Aus | Aus | Aus | Aus | An  | Aus | Aus | An  | Aus
An  | An  | Aus | Aus | Aus | An  | An  | Aus | Aus | An  | Aus
An  | An  | Aus | Aus | An  | Aus | An  | Aus | Aus | An  | An
An  | An  | Aus | Aus | An  | An  | An  | Aus | Aus | An  | An
An  | An  | Aus | An  | Aus | Aus | An  | Aus | An  | Aus | Aus
An  | An  | Aus | An  | Aus | An  | An  | Aus | An  | Aus | Aus
An  | An  | Aus | An  | An  | Aus | An  | Aus | Aus | An  | An
An  | An  | Aus | An  | An  | An  | An  | Aus | Aus | An  | An
An  | An  | An  | Aus | Aus | Aus | An  | Aus | Aus | An  | Aus
An  | An  | An  | Aus | Aus | An  | An  | Aus | Aus | An  | Aus
An  | An  | An  | Aus | An  | Aus | An  | Aus | Aus | An  | An
An  | An  | An  | Aus | An  | An  | An  | Aus | Aus | An  | An
An  | An  | An  | An  | Aus | Aus | An  | Aus | An  | Aus | Aus
An  | An  | An  | An  | Aus | An  | An  | Aus | An  | Aus | Aus
An  | An  | An  | An  | An  | Aus | An  | Aus | Aus | An  | An
An  | An  | An  | An  | An  | An  | An  | Aus | Aus | An  | An
```

Quellcode

```
import sys

# Textdatei einlesen
# sys.argv[1] ist das erste Kommandozeilenargument
with open(sys.argv[1]) as f:
    input_lines = f.read().split("\n")

class Layer:
    """
    Klasse zum Speichern einer Schicht in einer Konstruktion
    """

    def __init__(self, aufbau : list):
        self.aufbau = aufbau
```

```
    # Als nächstes kommen, Funktionen, die definieren, wie die einzelnen Bausteine bzw.
Komponenten mit Signalen umgehen
    # Jede Funktion repräsentiert die Funktionsweise einer Komponente
    # Alle Funktionen funktionieren dabei nach dem gleichen Prinzip:
    # Die ersten beiden Elemente werden aus inputs entnommen bzw. entfernt und als
Signale verarbeitet
    # Die verarbeiteten Signale werden dann an outputs angehängt

    def X(self, inputs, outputs):
        outputs.append(False)
        return inputs[1:], outputs

    def WW(self, inputs, outputs):
        sensor1, sensor2 = inputs[:2]
        outputs = outputs + [not (sensor1 and sensor2), not (sensor1 and sensor2)]
        return inputs[2:], outputs

    def BB(self, inputs, outputs):
        sensor1, sensor2 = inputs[:2]
        outputs = outputs + [sensor1, sensor2]
        return inputs[2:], outputs

    def Rr(self, inputs, outputs):
        sensor1, sensor2 = inputs[:2]
        outputs = outputs + [not sensor1, not sensor1]
        return inputs[2:], outputs

    def rR(self, inputs, outputs):
        sensor1, sensor2 = inputs[:2]
        outputs = outputs + [not sensor2, not sensor2]
        return inputs[2:], outputs

    def forward(self, inputs):
        """
        Wenn die Schicht weder Eingabeschicht noch Ausgabeschicht ist, dann kann mit
dieser Funktion der Input der vorherigen Schicht eingelesen und verarbeitet werden

        Returns:
            list: Die Ausgaben der Schicht, die von der nächsten Schicht eingelesen
werden können
        """
        outputs = []
        for component in self.aufbau:
            if component == "X":
                inputs, outputs = self.X(inputs, outputs)
            if component == "WW":
                inputs, outputs = self.WW(inputs, outputs)
            if component == "BB":
                inputs, outputs = self.BB(inputs, outputs)
            if component == "Rr":
                inputs, outputs = self.Rr(inputs, outputs)
            if component == "rR":
                inputs, outputs = self.rR(inputs, outputs)
        return outputs

    def input_layer(self, inputs):
        """
```

```
        Wenn die Schicht Eingabeschicht (die mit den Taschenlampen) ist, dann können
mit dieser Funktion die Zustände der Taschenlampen eingelesen werden

        Returns:
            list: Die Zustände der Taschenlampen in einer Form, die von der nächsten
Schicht eingelesen werden kann
        """
        outputs = []
        for component in self.aufbau:
            if component == "X":
                outputs.append(False)
            if "Q" in component:
                q_index = int(component[1:])-1
                outputs.append(inputs[q_index])
        return outputs

    def output_layer(self, inputs):
        """
        Wenn die Schicht Ausgabeschicht (die mit den LEDs) ist, dann kann mit dieser
Funktion die Ausgabe der vorherigen Schicht eingelesen werden

        Returns:
            list: Die Zustände der LEDs
        """
        outputs = []
        for component in self.aufbau:
            if "L" in component:
                l_index = int(component[1:])-1
                while len(outputs) < l_index + 1:
                    outputs.append(None)
                outputs[l_index] = inputs.pop(0)
            else:
                inputs.pop(0)
        return outputs

# Parameter einlesen:
line = input_lines.pop(0).split(" ")
n = int(line[0])
m = int(line[1])

# Konstruktion einlesen:
konstruktion = []
num_taschenlampen = 0
num_leds = 0
for i in range(m):
    line = input_lines.pop(0).replace("  ", " ").split(" ")[:n]
    aufbau = []
    while line != []:
        component = line.pop(0)
        if "Q" in component:
            num_taschenlampen += 1
        elif "L" in component:
            num_leds += 1
        elif not component == "X":
            component += line.pop(0)
        aufbau.append(component)
    konstruktion.append(Layer(aufbau))
```

```
# Alle möglichen Kombinationen der Taschenlampen (ein / aus) ermitteln:
kombinationen = [[]]
for t in range(num_taschenlampen):
    for k in list(kombinationen):
        kombinationen.remove(k)
        kombinationen.append(k+[False])
        kombinationen.append(k+[True])

# Für jede Kombination der Taschenlampen (ein / aus) die Zustände der LEDs (ein / aus)
ermitteln:
leds = []
for k in kombinationen:
    output = konstruktion[0].input_layer(k)
    for layer in konstruktion[1:-1]:
        output = layer.forward(output)
    output = konstruktion[-1].output_layer(output)
    leds.append(output)

# Ermittelte Zustände der LEDs in Tabellenform ausgeben:
zustände = {True:"An", False:"Aus"}
print("   | ".join(
    [f"Q{q+1}" for q in range(num_taschenlampen)]+
    [f"L{l+1}" for l in range(num_leds)]
))
for i in range(len(kombinationen)):
    print("| ".join(
        ["{:<5}".format(zustände[kombinationen[i][q]]) for q in
range(num_taschenlampen)]+
        ["{:<5}".format(zustände[leds[i][l]]) for l in range(num_leds)]
    ))
```