# Aufgabe 4: Nandu

Team-ID: 00112

Team-Name: 10m Dürer gym

Bearbeiter/-innen dieser Aufgabe:
Finn Degen

19. November 2023

## Inhaltsverzeichnis



## 1 Lösungsidee

### 1.1 Identifikation der Blocktypen

Um die Aufgabe zu lösen müssen wir erstmal definieren, was die einzelnen Blöcke genau machen.

- *weißer Block*:

| IN1 | IN2 | OUT1 | OUT2 |
|---|---|---|---|
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |

⇒ Wenn wir IN1, IN2 und ein beliebiges OUT betrachten, erinnert es stark an eine AND Wahrheitstabelle, nur ist OUT eben umgekehrt ⇒ OUT1 = OUT2 = NOT(AND(IN1,IN2)) = NAND(IN1,IN2)

- *roter Block*:

| IN1 | OUT1 | OUT2 |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 0 |

⇒ Wenn wir IN1 und ein beliebiges OUT betrachten, sieht man den NOT zusammenhand ⇒ OUT1 = OUT2 = NOT(IN1)

- *blauer Block*:

| IN1 | IN2 | OUT1 | OUT2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

⇒ Wenn wir IN1, IN2, OUT1 und OUT2 betrachten, sieht man das 1 und 2 jeweils gleich sind ⇒ OUT1 = IN1; OUT2 = IN2

## 1.2 Iteratives Berechnen der Zustände

Nun brauchen wir nur noch eine Methode, diese Blöcke in Gittern einzulesen und die Zustände zu berechnen. Dazu nehmen wir ein 2D Array *CALC_ TABLE*, in dem wir die Zustände für jede Koordinate des Gitters speichern.

> **Beispiel 1.** *Für ein 3x3 Gitter sieht das dann so aus:*
> *CALC_ TABLE = [ [0,0,0], [0,0,0], [0,0,0] ]*

Als ersten Schritt finden wir dann die Koordinaten unserer Input-Licht Blöcke (Q) und setzen diese in *CALC_ TABLE* auf die gewünschten Wahrheitswerte. Dann gehen wir durch jede Koordinate des Gitters: erst link nach rechts, dann oben nach unten und setzen den geradigen Wahrheitswert zu der spezifischen Block Funktion (siehe oben + Q + L nimmt Input von Reihe davor auf). Die Inputs nehmen wir von den Koordinaten eine Reihe davor. Um die richtige Blockfunktion zu finden, brauchen wir noch ein 2D Array *BLOCK_ TABLE*, in dem wir die Blöcke (W,R,r,B,Q,L) speichern.

> **Beispiel 2.** *Für ein 3x3 Gitter mit einem eingeschalteten Licht in der Mitte oben und einem blauem Block darunter sieht das dann so aus:*
> *Iteration 1:*
> *CALC_ TABLE = [ [0,1,0], [0„0], [0,0,0] ]*
> *BLOCK_ TABLE = [ [0,Q,0], [B,B,0], [0,0,0] ]*
> *Iteration 2:*
> *CALC_ TABLE = [ [0,1,0], [0,1,0], [0,0,0] ]*
> *BLOCK_ TABLE = [ [0,Q,0], [B,B,0], [L,L,0] ]*

Nach der letzten Iteration haben wir dann die Lösung für das Gitter in *CALC_ TABLE* gespeichert und können dann die Zustände aller Koordinaten ausgeben, die einen Output-Licht Block (L) haben.

# 2 Umsetzung

Der folgende Pseudocode beschreibt den Lösungsalgorithmus für die Aufgabe.
Zuerst werden folgende Variablen initialisiert:

```
CALC_TABLE ← 2D Array mit der der Größe des Gitters
BLOCK_TABLE ← AufgabeEinlesenZuBlockArray()
INPUT_COORDINATES ← FindeInputKoordinaten()
for Koordinate ∈ INPUT_COORDINATES do
    CALC_TABLE[Koordinate] ← Wahrheitswert des Input-Lichts
end for
OUTPUT_COORDINATES ← FindeOutputKoordinaten()
```

Dann wird das Gitter iterativ durchgegangen und die Zustände berechnet:

```
for Zeile ∈ CALC_TABLE do
    for Koordinate ∈ Zeile do
        CALC_TABLE[Koordinate] ← BerechneBlock(CALC_TABLE[Koordinate], CALC_TABLE, Koordinate)
    end for
end for
```

Die Methode *BerechneBlock* berechnet den Zustand einer Koordinate anhand der Blockfunktion und den Inputs. Die Blockfunktion ordnet sie anhand des geradigen und des letzen Blockcharakters aus *BLOCK_ TABLE* zu, da die großen Blöcke ja 2 Einheiten lang sind. Die Inputs nimmt sie von den Koordinaten eine Reihe davor. Sie wird später in Abschnitt 4 noch genauer erläutert.

Zum Schluss werden die Zustände der Output-Lichter ausgegeben:

```
for Koordinate ∈ OUTPUT_COORDINATES do
    Ausgabe(CALC_TABLE[Koordinate])
end for
```

# 3 Beispiele

Im folgenden wird das Programm mit allen Beispielaufgaben ausgeführt

### Beispiel 1

```
Q1: False Q2: False L1: True L2: True
Q1: False Q2: True L1: True L2: True
Q1: True Q2: False L1: True L2: True
Q1: True Q2: True L1: False L2: False
```

### Beispiel 2

```
Q1: False Q2: False L1: False L2: True
Q1: False Q2: True L1: False L2: True
Q1: True Q2: False L1: False L2: True
Q1: True Q2: True L1: True L2: False
```

### Beispiel 3

```
Q1: False Q2: False Q3: False L1: True L2: False L3: False L4: True
Q1: False Q2: False Q3: True L1: True L2: False L3: False L4: False
Q1: False Q2: True Q3: False L1: True L2: False L3: True L4: True
Q1: False Q2: True Q3: True L1: True L2: False L3: True L4: False
Q1: True Q2: False Q3: False L1: False L2: True L3: False L4: True
```

```
Q1: True Q2: False Q3: True L1: False L2: True L3: False L4: False
Q1: True Q2: True Q3: False L1: False L2: True L3: True L4: True
Q1: True Q2: True Q3: True L1: False L2: True L3: True L4: false
```

**Beispiel 4**

```
Q1: False Q2: False Q3: False Q4: False L1: False L2: False
Q1: False Q2: False Q3: False Q4: True L1: False L2: False
Q1: False Q2: False Q3: True Q4: False L1: False L2: True
Q1: False Q2: False Q3: True Q4: True L1: False L2: False
Q1: False Q2: True Q3: False Q4: False L1: True L2: False
Q1: False Q2: True Q3: False Q4: True L1: True L2: False
Q1: False Q2: True Q3: True Q4: False L1: True L2: True
Q1: False Q2: True Q3: True Q4: True L1: True L2: False
Q1: True Q2: False Q3: False Q4: False L1: False L2: False
Q1: True Q2: False Q3: False Q4: True L1: False L2: False
Q1: True Q2: False Q3: True Q4: False L1: False L2: True
Q1: True Q2: False Q3: True Q4: True L1: False L2: False
Q1: True Q2: True Q3: False Q4: False L1: False L2: False
Q1: True Q2: True Q3: False Q4: True L1: False L2: False
Q1: True Q2: True Q3: True Q4: False L1: False L2: True
Q1: True Q2: True Q3: True Q4: True L1: False L2: False
```

**Beispiel 5** Achtung: Hier habe ich Absätze machen müssen, weil der Platz sonst nicht reicht

```
Q1: False Q2: False Q3: False Q4: False Q5: False Q6: False L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: False Q3: False Q4: False Q5: False Q6: True L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: False Q3: False Q4: False Q5: True Q6: False L1: False L2: False L3: False
    L4: True L5: True
Q1: False Q2: False Q3: False Q4: False Q5: True Q6: True L1: False L2: False L3: False
    L4: True L5: True
Q1: False Q2: False Q3: False Q4: True Q5: False Q6: False L1: False L2: False L3: True
    L4: False L5: False
Q1: False Q2: False Q3: False Q4: True Q5: False Q6: True L1: False L2: False L3: True L4
    : False L5: False
Q1: False Q2: False Q3: False Q4: True Q5: True Q6: False L1: False L2: False L3: False
    L4: True L5: True
Q1: False Q2: False Q3: False Q4: True Q5: True Q6: True L1: False L2: False L3: False L4
    : True L5: True
Q1: False Q2: False Q3: True Q4: False Q5: False Q6: False L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: False Q3: True Q4: False Q5: False Q6: True L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: False Q3: True Q4: False Q5: True Q6: False L1: False L2: False L3: False
    L4: True L5: True
Q1: False Q2: False Q3: True Q4: False Q5: True Q6: True L1: False L2: False L3: False L4
    : True L5: True
Q1: False Q2: False Q3: True Q4: True Q5: False Q6: False L1: False L2: False L3: True L4
    : False L5: False
Q1: False Q2: False Q3: True Q4: True Q5: False Q6: True L1: False L2: False L3: True L4:
     False L5: False
Q1: False Q2: False Q3: True Q4: True Q5: True Q6: False L1: False L2: False L3: False L4
    : True L5: True
Q1: False Q2: False Q3: True Q4: True Q5: True Q6: True L1: False L2: False L3: False L4:
     True L5: True
Q1: False Q2: True Q3: False Q4: False Q5: False Q6: False L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: True Q3: False Q4: False Q5: False Q6: True L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: True Q3: False Q4: False Q5: True Q6: False L1: False L2: False L3: False
    L4: True L5: True
Q1: False Q2: True Q3: False Q4: False Q5: True Q6: True L1: False L2: False L3: False L4
    : True L5: True
```

```
Q1: False Q2: True Q3: False Q4: True Q5: False Q6: False L1: False L2: False L3: True L4
    : False L5: False
Q1: False Q2: True Q3: False Q4: True Q5: False Q6: True L1: False L2: False L3: True L4:
     False L5: False
Q1: False Q2: True Q3: False Q4: True Q5: True Q6: False L1: False L2: False L3: False L4
    : True L5: True
Q1: False Q2: True Q3: False Q4: True Q5: True Q6: True L1: False L2: False L3: False L4:
     True L5: True
Q1: False Q2: True Q3: True Q4: False Q5: False Q6: False L1: False L2: False L3: False
    L4: True L5: False
Q1: False Q2: True Q3: True Q4: False Q5: False Q6: True L1: False L2: False L3: False L4
    : True L5: False
Q1: False Q2: True Q3: True Q4: False Q5: True Q6: False L1: False L2: False L3: False L4
    : True L5: True
Q1: False Q2: True Q3: True Q4: False Q5: True Q6: True L1: False L2: False L3: False L4:
     True L5: True
Q1: False Q2: True Q3: True Q4: True Q5: False Q6: False L1: False L2: False L3: True L4:
     False L5: False
Q1: False Q2: True Q3: True Q4: True Q5: False Q6: True L1: False L2: False L3: True L4:
    False L5: False
Q1: False Q2: True Q3: True Q4: True Q5: True Q6: False L1: False L2: False L3: False L4:
     True L5: True
Q1: False Q2: True Q3: True Q4: True Q5: True Q6: True L1: False L2: False L3: False L4:
    True L5: True
Q1: True Q2: False Q3: False Q4: False Q5: False Q6: False L1: True L2: False L3: False
    L4: True L5: False
Q1: True Q2: False Q3: False Q4: False Q5: False Q6: True L1: True L2: False L3: False L4
    : True L5: False
Q1: True Q2: False Q3: False Q4: False Q5: True Q6: False L1: True L2: False L3: False L4
    : True L5: True
Q1: True Q2: False Q3: False Q4: False Q5: True Q6: True L1: True L2: False L3: False L4:
     True L5: True
Q1: True Q2: False Q3: False Q4: True Q5: False Q6: False L1: True L2: False L3: True L4:
     False L5: False
Q1: True Q2: False Q3: False Q4: True Q5: False Q6: True L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: False Q3: False Q4: True Q5: True Q6: False L1: True L2: False L3: False L4:
     True L5: True
Q1: True Q2: False Q3: False Q4: True Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: False Q3: True Q4: False Q5: False Q6: False L1: True L2: False L3: False L4
    : True L5: False
Q1: True Q2: False Q3: True Q4: False Q5: False Q6: True L1: True L2: False L3: False L4:
     True L5: False
Q1: True Q2: False Q3: True Q4: False Q5: True Q6: False L1: True L2: False L3: False L4:
     True L5: True
Q1: True Q2: False Q3: True Q4: False Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: False Q3: True Q4: True Q5: False Q6: False L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: False Q3: True Q4: True Q5: False Q6: True L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: False Q3: True Q4: True Q5: True Q6: False L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: False Q3: True Q4: True Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: False Q4: False Q5: False Q6: False L1: True L2: False L3: False L4
    : True L5: False
Q1: True Q2: True Q3: False Q4: False Q5: False Q6: True L1: True L2: False L3: False L4:
     True L5: False
Q1: True Q2: True Q3: False Q4: False Q5: True Q6: False L1: True L2: False L3: False L4:
     True L5: True
Q1: True Q2: True Q3: False Q4: False Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: False Q4: True Q5: False Q6: False L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: True Q3: False Q4: True Q5: False Q6: True L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: True Q3: False Q4: True Q5: True Q6: False L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: False Q4: True Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: True Q4: False Q5: False Q6: False L1: True L2: False L3: False L4:
```

```
     True L5: False
Q1: True Q2: True Q3: True Q4: False Q5: False Q6: True L1: True L2: False L3: False L4:
    True L5: False
Q1: True Q2: True Q3: True Q4: False Q5: True Q6: False L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: True Q4: False Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: True Q4: True Q5: False Q6: False L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: True Q3: True Q4: True Q5: False Q6: True L1: True L2: False L3: True L4:
    False L5: False
Q1: True Q2: True Q3: True Q4: True Q5: True Q6: False L1: True L2: False L3: False L4:
    True L5: True
Q1: True Q2: True Q3: True Q4: True Q5: True Q6: True L1: True L2: False L3: False L4:
    True L5: True
```

---

# 4 Quellcode

Wir fangen mit der Initialisierung der Variablen wie in Abschnitt 2 an:

```python
table: list[list[str]] = []  # Pseudocode: BLOCK_TABLE
for row in raw_data:  # Pseudocode: AufgabeEinlesenZuBlockArray
    table.append(row.split())

# initialize Qs and Ls
num_q = 0
indecies_q = []
indecies_l = []
for row_index, _ in enumerate(table):
    for column_index, element in enumerate(table[row_index]):  # first row
        if element.startswith("Q"):
            num_q += 1
            indecies_q.append((row_index, column_index))
        elif element.startswith("L"):
            indecies_l.append((row_index, column_index))
```

Da wir alle möglichen Input-Licht Kombinationen ausgeben müssen, brauchen wir eine Methode, die uns alle Kombinationen von 0 und 1 mit einer bestimmten Länge ausgibt. Dazu benutzen wir die *itertools* Bibliothek.

```python
possible_q_combinations = list(itertools.product([False, True], repeat=num_q))
```

Nun implementieren wir die Haupt Algorithmus, dies ist der selbe wie in Abschnitt 2 nur wird er n mal für alle permutationen der Input-Lichter ausgeführt

```python
for possible_q_combination in possible_q_combinations:
  # this table saves the booleans for the light outputs
  # Psuedocode: CALC_TABLE
  calculation_table = [[None for _ in range(len(table[0]))] for _ in range(len(table))]

  # print Qs
  for num_of_possible_q, possible_q_value in enumerate(possible_q_combination):
      # start printing table
      print(f"Q{num_of_possible_q+1}: {possible_q_value}", end=" ")
      # add the current value of true or false for
      # each Q so that we have all combinations in the end
      calculation_table[indecies_q[num_of_possible_q][0]][
          indecies_q[num_of_possible_q][1]
      ] = possible_q_combination[num_of_possible_q]

  # go trough each row
  # and compute the bool values for the light outputs
  for row_index, _ in enumerate(table):
      last_element = None # last element has to be reset, else it will think that blocks on the edge sp
      for column_index, element in enumerate(table[row_index]):
          # Pseudocode: BerechneBlock
```

```
            last_element = calculate_element(element, last_element, calculation_table, row_index, column_
23
    # print Ls
25  for num_current_l, (row_index, column_index) in enumerate(indecies_l):
        print(
27          f"L{num_current_l+1}: {calculation_table[row_index][column_index]}",
            end=" ",
29      )
```

Nun schauen wir uns nochmal die Methode *calculate_element* an, die die neuen Zustände der Blöcke berechnet.

```
def calculate_element(element: str, last_element: str, calculation_table, row_index, column_index):
2 """Berechnet den Wahrheitswehrt für das gegebene Element mit der zusätzlichen Information des letzten E

4 :param element: Das aktuelle Element
  :param last_element: Das letzte Element
6 :param calculation_table: Die Tabelle mit den Wahrheitswerten. Wird direkt verändert
  :param row_index: Die aktuelle Zeile
8 :param column_index: Die aktuelle Spalte
  :returns: Das neue letzte Element
10 """
  if last_element:
12     match element:
           case "W":
14             if last_element == "W":
                   calculation_table[row_index][column_index] = calculation_table[row_index][column_index
16                     calculation_table[row_index - 1][column_index]
                       and calculation_table[row_index - 1][column_index - 1]
18                 )  # NAND
                   element = ""
20         case "R":
               if last_element == "r":
22                 calculation_table[row_index][column_index] = calculation_table[row_index][column_index
                       calculation_table[row_index - 1][column_index]
24                 )
                   # NOT
26                 element = ""
           case "r":
28             if last_element == "R":
                   calculation_table[row_index][column_index] = calculation_table[row_index][column_index
30                     calculation_table[row_index - 1][column_index - 1]
                   )
32                 # NOT
                   element = ""
34         case "B":
               if last_element == "B":
36                 calculation_table[row_index][column_index] = calculation_table[row_index - 1][column_in
                   calculation_table[row_index][column_index - 1] = calculation_table[row_index - 1][colum
38                 # EQUAL
                   element = ""
40
  if element.startswith("L"):
42     calculation_table[row_index][column_index] = calculation_table[row_index - 1][column_index]
       # EQUAL
44 return element
```

Diese Methode findet doppel Blöcke anhand des letzen Elements und wendet so die Blockfunktion an dem CALC_TABLE an.
Für erfolgreich gefundene Blöcke wird das geradige Element auf einen leeren String gesetzt, damit die Methode WWWW,BBBB,... nicht als 3 Blöcke, sondern als 2 Blöcke erkennt.
Die Methode gibt dann das geradie Element zurück, damit es als letztes Element für das nächste Aufrufen der Funktion gespeichert werden kann.