Aufgabe 4: Nandu

Team-ID: 00879

Bearbeiter/-in dieser Aufgabe:
Karl Zschiebsch

12. November 2023

Inhaltsverzeichnis

Lösungsidee

Das Licht wird von den Taschenlampen über die Bausteine zu der Ausgabe durchgegeben. Hierbei wird jeder Baustein als Blöcke betrachtet. Die Blöcke sind jeweils mit einem anderen Block verknüpft. Die Taschenlampen werden auf als Eingabe betrachtet, während die letzte Reihe als Ausgabe betrachtet wird. Es werden alle Permutationen berechnet, die alle möglichen Zustände für die Eingabereihe darstellen. Für jede Permutation wird die Eingabereihe dementsprechend gesetzt. Von der Eingabereihe ausgehend wird für jeden Block bestimmt, ob er Licht zur nächsten Reihe weitergibt oder nicht. Falls er Licht weitergibt, wird ein Wert für den Sensor des Blockes in der darunter liegenden Reihe entsprechend gesetzt. Dieses Vorgehen wird wiederholt, bis die letzte Ausgabereihe erreicht ist. Alle Zustände der Eingabe und Ausgabequellen wird ausgelesen. Danach werden die Zustände des Feldes zurückgesetzt. Dieses Verfahren wird für alle Permutationen wiederholt.

Umsetzung

Beispiele

Dies sind die Lösungen zu den ersten drei Beispielen der Website. Nandu0.txt ist einfach nur das Beispiel aus dem Aufgabenblatt, Abbildung Links.

```
nandu0.txt
Q1  Q2  | L1  L2
---------+------------
0   0   | 0   0
1   0   | 0   0
```

```
1   1   |  1   1
0   1   |  0   0

nandu1.txt
Q1   Q2  |  L1   L2
----------+------------
0   0   |  1   1
1   0   |  1   1
1   1   |  0   0
0   1   |  1   1

nandu2.txt
Q1   Q2  |  L1   L2
----------+------------
0   0   |  0   1
0   1   |  0   1
1   0   |  0   1
1   1   |  1   0

nandu3.txt
Q1   Q2   Q3  |  L1   L2   L3   L4
--------------+---------------------
0   0   0   |  1   0   0   1
0   0   1   |  1   0   0   0
0   1   0   |  1   0   1   1
0   1   1   |  1   0   1   0
1   0   0   |  0   1   0   1
1   0   1   |  0   1   0   0
1   1   0   |  0   1   1   1
1   1   1   |  0   1   1   0
```

Quellcode

Dies ist der Quellcode, geschrieben in Python. Es wird itertools importiert, um alle Permutationen der Zustände der Taschenlampen zu berechnen.

```python
import itertools


class Position:
    def __init__(self, x: int, y: int):
        self.x = x
        self.y = y

    def __add__(self, other) -> 'Position':
        return Position(self.x + other.x, self.y + other.y)


class Environment:
    def __init__(self, path: str):
        self.fields: list[list['Block']] = []
```

```
    self.sources = []
    self.results = []
    with open(path, 'r') as reader:
        self.n, self.m = [int(v) for v in reader.readline().replace('\n', '').split(' ')]
        for y in range(self.m):
            self.fields.append([])
            types = reader.readline().replace('\n', '').replace('  ', ' ').split(' ')
            last = None
            for x in range(self.n):
                block = Block(self, Position(x, y), types[x].strip())
                if block.requires_connection():
                    if last is None:
                        last = block
                    else:
                        block.con = last
                        last.con = block
                        last = None
                elif block.is_source():
                    self.sources.append(block)
                elif block.is_result():
                    self.results.append(block)
                self.fields[y].append(block)

def deactivate_all(self) -> None:
    for array in self.fields[1:]:
        for field in array:
            field.activated = False

def reactivate_all(self) -> None:
    for array in self.fields[:-1]:
        for field in array:
            field.process_activation()

def get_header(self) -> str:
    build = ""
    for source in self.sources:
        build += f"{source.type:<5}"
    build += '|   '
    for result in self.results:
        build += f"{result.type:<5}"
    temp = build
    build += '\n'
    for char in temp:
        build += '+' if char == '|' else '-'
    return build

def get_state(self) -> str:
    build = ""
    for source in self.sources:
        build += f"{source.activated:<5}"
    build += '|   '
    for result in self.results:
        build += f"{result.activated:<5}"
    return build
```

```
def white_activation(white: 'Block') -> bool:
    return not (white.is_activated() and white.con.is_activated())


def red_major_activation(red_major: 'Block') -> bool:
    return not red_major.is_activated()


def red_minor_activation(red_minor: 'Block') -> bool:
    return not red_minor.con.is_activated()


def blue_activation(blue: 'Block') -> bool:
    return blue.is_activated()


class Block:
    activation_map = {
        'W': white_activation,
        'R': red_major_activation,
        'r': red_minor_activation,
        'B': blue_activation
    }

    def __init__(self, env: Environment, pos: Position, t: str, con: 'Block' = None):
        self.env = env
        self.pos = pos
        self.con = con
        self.type = t
        self.activated = False

    def is_activated(self) -> bool:
        return self.activated

    def activates(self) -> bool:
        func = Block.activation_map.get(self.type)
        if func is None:
            raise ValueError(f"Type : {self.type} ? {self.is_source()}")
        return func(self)

    def process_activation(self):
        if self.is_empty():
            return
        block = self.env.fields[self.pos.y + 1][self.pos.x]
        if self.is_source():
            block.activated = self.activated
        elif self.requires_connection():
            if self.activates():
                block.activated = True
        else:
            raise ValueError(self.type)

    def requires_connection(self) -> bool:
        return self.is_white() or self.is_blue() or self.is_red()
```

```
    def get_index(self) -> int:
        return int(self.type[1])

    def is_source(self) -> bool:
        return self.type[0] == 'Q'

    def is_result(self) -> bool:
        return self.type[0] == 'L'

    def is_white(self) -> bool:
        return self.type[0] == 'W'

    def is_blue(self) -> bool:
        return self.type[0] == 'B'

    def is_red(self) -> bool:
        return self.type.capitalize()[0] == 'R'

    def is_empty(self) -> bool:
        return self.type[0] == 'X'


environment = Environment(input("Please enter file: "))

permutation = list(itertools.product(range(2), repeat=len(environment.sources)))
print(environment.get_header())
for i in permutation:
    environment.deactivate_all()
    for j in range(len(environment.sources)):
        environment.sources[j].activated = i[j] == 1
    environment.reactivate_all()
    print(environment.get_state())
```