Aufgabe A4: Nandu

Team-ID: 00250

pip install BWINF_42

Bearbeiter dieser Aufgabe: Imanuel Fehse (Teilnahme-ID: 69274)

01.11.2023

Inhaltsverzeichnis

1 Aufgabe

Schreibe ein Programm, das eine Beschreibung einer Konstruktion einliest und angibt, wie die LEDs der ganz rechts liegenden Bausteine auf die Zustände der Lichtquellen reagieren.

2 Lösungsidee

Jeder der Bausteine wird durch ein Objekt repräsentiert, dass sich in einer zweidimensionalen Liste befindet, um die Konstruktion zu repräsentieren. Unterschiedliche Arten von Bausteinen haben jeweils ihr eigenes Objekt, das auf das Verhalten und das Aussehen der Bausteinsorte angepasst ist.

Nachdem diese zweidimensionale Liste erstellt wurde, also die Eingabedatei in eine für das Programm verwendbare übersetzt wurde, werden alle möglichen Kombinationen der Zustände,

in denen sich die Taschenlampen, die zur Eingabe genutzt werden, ermittelt und zwischengespeichert.
Anschließend ermittelt das Programm für jede der Eingabekombinationen die Zustände der Ausgänge, indem es von links nach rechts die Ausgaben der individuellen Bausteine berechnet, welche Abhängig von den Blöcken sind, die in der Reihe zuvor „geschalten“ sind.
Sind alle Eingabekombinationen abgearbeitet worden, so gibt das Programm eine Liste aus, die angibt, in welchem Zustand sich die verschiedenen Ausgänge für eine gegebene Eingabekombination befinden.

3 Umsetzung

3.1 Bausteinklassen

Zunächst wird für jeden Bausteintypen jeweils eine Klasse erstellt, von der aus später die Objekte erstellt werden, die die einzelnen Bausteine repräsentieren. Die Klassen werden jeweils auf den Bausteintypen spezialisiert. Was jedoch jede Klasse hat, die einen Bausteintypen repräsentiert, ist eine Typenvariable[1], die angibt, um was für einen Bausteintypen es sich handelt.
Bausteintypen-Klassen, die eine Lampe (Ein- & Ausgabe) repräsentieren, haben eine Variable, die den Status der Lampe angibt, also ob sie angeschaltet ist oder nicht. Lampen werden im Programm ebenfalls als Baustein angesehen, um das Programm zu vereinfachen.
Bausteintypen-Klassen, die keine Lampen sind, also alle anderen Bausteine haben eine Funktion, die die Eingabe in einen Baustein (= wird der Sensor des Bausteins bestrahlt oder nicht?) einließt und daraus die Ausgabe des Bausteins ermittelt. Die ermittelte Ausgabe wird dann auf dann auf die Ausgabevariablen des Bausteins geschrieben.

3.2 Algorithmus zur Erstellung der Konstruktion

Nachdem das Programm über die verschiedenen Bausteinklassen verfügt, wird ein weiteres Objekt von einer Klasse erzeugt, das die Konstruktion beinhaltet und verwaltet. Bei der Initialisierung dieses Objekts wird die Eingabedatei eingelesen und alle nötigen Speichervariablen erstellt, die die Klasse benötigt, um später eine Lösung zu finden.
Anschließend wird aus der Datei eine Konstruktion erstellt, die das Programm verwenden kann. In der Aufgabenstellung werden die Konstruktionen von links nach rechts gebaut. Da es für das Ergebnis irrelevant ist, ob die Konstruktion von links nach rechts oder von oben nach unten gebaut wird (zweidimensional gesehen), erstellt der Algorithmus eine zweidimensionale Liste,

[1] Wird später im Programm verwendet, um festzustellen, um welchen Bausteintypen es sich handelt, dass der Algorithmus allgemein für alle Bausteintypen geschrieben ist.

die die Konstruktion von oben nach unten aufbaut. Dieser Algorithmus (Implementierung[2]) wird durch den folgenden Pseudocode dargestellt:

```
1  Create list in which the construction will be saved
2  FOR every line in input file:
3      Create list in which the construction of the current line is saved
4      FOR every block in line
5          IF field is empty:
6              Add empty field to current line at current position
7          ELIF field is a lamp input block:
8              Add lamp input object to current line at current position
9          ELIF field is a lamp output block:
10             Add lamp output object to current line at current position
11         ELIF field is a white block:
12             Add white brick object to current line at current position
13             Add the same object to the next field (block is 2 fields wide)
14             Skip 1 field
15         ELIF field is a blue block:
16             Add blue brick object to current line at current position
17             Add the same object to the next field (block is 2 fields wide)
18             Skip 1 field
19         ELIF field is a red block:
20             IF the sensor of the red block is at current position:
21                 Add red brick object to current line at current point
22                 Add the same object to the next field (block is 2 fields wide)
23                 Set the sensor position to the left side
24                 Skip 1 field
25             ELSE:
26                 Add red brick object to current line at current point
27                 Add the same object to the next field (block is 2 fields wide)
28                 Set the sensor position to the right side
29                 Skip 1 field
30         Go to next field in line
31     Add current line to construction
```

Der Algorithmus bearbeitet die Eingabedatei Zeile für Zeile (Y-Achse der Konstruktion) und durchläuft jede Zeile von links nach rechts (X-Achse der Konstruktion). Der Algorithmus prüft für jedes Feld, um welche Art von Baustein es sich in dem Feld handelt.

Bausteine, die ein Feld breit sind, sind leere Felder genauso wie Ein- und Ausgabelampen. Jenachdem, was sich im aktuellen Feld befindet, fügt der Algorithmus einen dieser Bausteine in dem entsprechenden Feld ein.

Bausteine, die zwei Felder breit sind, sind weiße, blaue und rote Bausteine. Bei weißen und blauen Bausteinen fügt der Algorithmus dasselbe Objekt für das aktuelle Feld und das nächste Feld ein. Da dadurch schon bekannt ist, was sich im nächsten Feld befindet, wird das nächste Feld übersprungen. Befindet sich in dem Feld ein roter Baustein, so ermittelt der Algorithmus,

[2] Implementierung des Algorithmus zur Erstellung der Konstruktion: hier

ob sich der Sensor des roten Bausteins im aktuellen Feld oder im nächsten Feld befindet. Sobald die Position des Sensors ermittelt wurde, wird derselbe rote Baustein für das aktuelle und das nächste Feld eingefügt. Anschließend, wird die Position des Senors im Baustein gespeichert und das nächste Feld übersprungen, da schon bekannt ist, was sich in diesem Feld befindet.
Hat der Algorithmus alle Zeilen bearbeitet, ist die Konstruktion mit den Bausteinobjekten vollständig und kann für weitere Schritte verwendet werden.

3.3 Algorithmus zur Ermittlung der Ausgaben der Konstruktion für eine bestimmte Kombination der Eingaben

Als nächster Schritt werden alle Ein- und Ausgabelampen Objekte in der Konstruktion ermittelt und in jeweilige Listen eingefügt. Mit der Liste der Eingabelampen Objekte kann nun jede mögliche Kombination ermittelt werden, in denen sich die Eingabelampenobjekte befinden können. Hierfür wird ein boolescher Kombinationsgenerator genutzt (Implementierung[3]), der binäre Werte nutzt, um den Zustand einer Eingabelampe darzustellen (1 = an, 0 = aus).

3.4 Ermitteln der Ausgaben für alle Kombinationen an Eingaben

Nachdem alle Kombinationen der Eingabelampen ermittelt wurden, wird ein weiterer Algorithmus verwendet (Implementierung[4]), um die Zustände der Ausgänge für jede Eingabekombination zu ermitteln, welcher durch den folgenden Pseudocode dargestellt wird:

```
1  FOR every input combination:
2      Set states of input lamps according to current input combination
3      FOR every line in construction (except for first line → only input lamps in this line):
4          FOR every field in line:
5              IF current field is empty:
6                  Go to next field
7              ELIF current field is red brick:
8                  IF current field is the position of the sensor in the brick:
9                      IF output if brick in line above is 1:
10                         Set output of red brick to 0
11                     ELSE:
12                         Set outputs of red brick to 1
13                 ELSE:
14                     Go to next field
15             ELIF current field is a white brick:
16                 IF at least one of the outputs of the bricks in the line above is 1:
17                     Set outputs of white brick to 1
18                 ELSE:
19                     Set outputs of white brick to 0
20             ELIF current field is a blue brick:
21                 IF output of the brick in the line above the current position is 1:
22                     Set the output of the blue brick at the current position to 1
```

[3] Implementierung des Kombinationsgenerators: hier
[4] Implementierung des Algorithmus zur Ermittlung der Ausgaben für alle Kombinationen an Eingaben: hier

```
                    ELSE:
                        Set output of the blue brick at current position to 0
                ELIF current field is a lamp output brick:
                    IF output of brick in line above is 1:
                        Set lamp output to 1
                    ELSE:
                        Set lamp output to 0
            Get states of all output lamps and add to list with the current input combination
```

Zunächst setzt der Algorithmus die Ausgänge auf die entsprechenden Zustände der aktuellen Eingabekombination. Anschließend durchläuft der Algorithmus die Konstruktion Zeile für Zeile und in jeder Zeile Feld nach Feld. Für jedes Feld wird überprüft, um welchen Bausteintypen es sich handelt.

Befindet sich im aktuellen Feld nichts oder eine Eingabelampe, so wird das Feld übersprungen. Handelt es sich jedoch um einen weißen, blauen oder roten Baustein, so wird zunächst ermittelt, ob die aktuelle Position die linke oder rechte Seite des Bausteins ist.

Handelt es sich um einen Roten Baustein, so wird überprüft, ob die aktuelle Position der Position des Sensors entspricht. Ist dies der Fall, so werden die Ausgänge des roten Bausteins auf die entsprechenden Zustände gesetzt. Ist der Eingang in den Sensor (entspricht dem Ausgang des Bausteins über dem Sensor des roten Bausteins) 1, so werden beide Ausgänge des roten Sensors auf 0 gesetzt. Ist der Eingang 0, so werden die die Ausgänge auf 1 gesetzt.

Handelt es sich bei dem aktuellen Baustein um einen weißen, so werden die Eingänge in die beiden Sensoren des weißen Bausteins ermittelt. Solange mindestens einer der Eingänge 1 ist, werden beide Ausgänge des weißen Sensors auf 1 gesetzt. Ist kein Eingang 1, so werden beide Ausgänge auf 0 gesetzt.

Ist der aktuelle Baustein ein blauer Baustein, so wird die aktuelle Seite des blauen Bausteins auf denselben Wert wie der Eingang des Sensors auf dieser Seite gesetzt, da der Sensor die Eingabe die er erhält einfach an den nächsten Sensor weitergibt.

Hat der Algorithmus einen Ausgabelampen Baustein erreicht, so wird der Zustand der Lampe auf den Zustand des Ausgangs des Bausteins im Feld über der Lampe gesetzt.

Nachdem der Algorithmus den Zustand aller Bausteine in der Konstruktion für eine Kombination ermittelt hat, werden die Zustände der Ausgabelampen Bausteine für die aktuelle Eingabekombination gespeichert.

Der Algorithmus wird für alle Eingabekombinationen wiederholt.

3.5 Speichern der Ergebnisse

Nachdem die Ausgabeergebnisse für alle Eingabekombinationen ermittelt wurden, werden die Ergebnisse in Tabellenform in einer Textdatei gespeichert. Die erste Zeile gibt an, um welche

Ein- oder Ausgabelampe es sich handelt. Alle weiteren Zeilen geben die Eingabekombinationen und ihre zugehörigen Ausgabeergebnisse an. (Implementierung[5])

[5] Implementierung der Speicherung der ermittelten Ergebnisse: hier

4 Beispiele

Nandu1.txt

```
Q1  Q2  L1  L2
 0   0   1   1
 0   1   1   1
 1   0   1   1
 1   1   0   0
```

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   1   1   1
 0   0   1   1   1   1   1
 0   1   0   0   1   0   1
 0   1   1   0   1   1   1
 1   0   0   1   0   1   0
 1   0   1   1   0   1   0
 1   1   0   0   0   0   0
 1   1   1   0   0   1   0
```

Nandu4.txt

```
Q1  Q2  Q3  Q4  L1  L2
 0   0   0   0   0   1
 0   0   0   1   0   1
 0   0   1   0   1   1
 0   0   1   1   1   1
 0   1   0   0   0   0
 0   1   0   1   0   1
 0   1   1   0   1   0
 0   1   1   1   1   1
 1   0   0   0   0   1
 1   0   0   1   0   1
 1   0   1   0   0   1
 1   0   1   1   0   1
 1   1   0   0   0   0
 1   1   0   1   0   1
 1   1   1   0   0   0
 1   1   1   1   0   1
```

Nandu5.txt

Q1	Q2	Q3	Q4	Q5	Q6	L1	L2	L3	L4	L5
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	1	0	0	1
0	0	0	0	1	1	0	1	0	0	1
0	0	0	1	0	0	0	1	0	1	0
0	0	0	1	0	1	0	1	0	1	0
0	0	0	1	1	0	0	1	0	0	1
0	0	0	1	1	1	0	1	0	0	1
0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0	0	1
0	0	1	1	0	0	0	1	0	1	0
0	0	1	1	0	1	0	1	0	1	0
0	0	1	1	1	0	0	1	0	0	1
0	0	1	1	1	1	0	1	0	0	1
0	1	0	0	0	0	0	1	0	0	0
0	1	0	0	0	1	0	1	0	0	0
0	1	0	0	1	0	0	1	0	0	1
0	1	0	0	1	1	0	1	0	0	1
0	1	0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	0	1	0	1	0
0	1	0	1	1	0	0	1	0	0	1
0	1	0	1	1	1	0	1	0	0	1
0	1	1	0	0	0	0	1	0	0	0
0	1	1	0	0	1	0	1	0	0	0
0	1	1	0	1	0	0	1	0	0	1
0	1	1	0	1	1	0	1	0	0	1
0	1	1	1	0	0	0	1	0	1	0
0	1	1	1	0	1	0	1	0	1	0
0	1	1	1	1	0	0	1	0	0	1
0	1	1	1	1	1	0	1	0	0	1
1	0	0	0	0	0	1	0	1	0	0
1	0	0	0	0	1	1	0	1	0	0
1	0	0	0	1	0	1	0	1	0	1
1	0	0	0	1	1	1	0	1	0	1
1	0	0	1	0	0	1	1	1	1	0
1	0	0	1	0	1	1	1	1	1	0
1	0	0	1	1	0	1	1	1	0	1
1	0	0	1	1	1	1	1	1	0	1
1	0	1	0	0	0	1	0	1	0	0
1	0	1	0	0	1	1	0	1	0	0
1	0	1	0	1	0	1	0	1	0	1
1	0	1	0	1	1	1	0	1	0	1
1	0	1	1	0	0	1	1	1	1	0
1	0	1	1	0	1	1	1	1	1	0
1	0	1	1	1	0	1	1	1	0	1
1	0	1	1	1	1	1	1	1	0	1
1	1	0	0	0	0	1	0	1	0	0
1	1	0	0	0	1	1	0	1	0	0
1	1	0	0	1	0	1	0	1	0	1

```
1   1   0   0   1   1   1   0   1   0   1
1   1   0   1   0   0   1   1   1   1   0
1   1   0   1   0   1   1   1   1   1   0
1   1   0   1   1   0   1   1   1   0   1
1   1   0   1   1   1   1   1   1   0   1
1   1   1   0   0   0   1   0   1   0   0
1   1   1   0   0   1   1   0   1   0   0
1   1   1   0   1   0   1   0   1   0   1
1   1   1   0   1   1   1   0   1   0   1
1   1   1   1   0   0   1   1   1   1   0
1   1   1   1   0   1   1   1   1   1   0
1   1   1   1   1   0   1   1   1   0   1
1   1   1   1   1   1   1   1   1   0   1
```

5 Wichtige Teile des Codes

```
 def create_construction_from_file(self):
    # for each line (y-value) of the construction
    for line in self.data[1:self.y+1]:
        line = line.split()
        current_line_construction = []
        # convert file to objects
        i = 0
        while i < self.x:
            # if position is empty
            if line[i] == "X":
                current_line_construction.append(None)
            # if position in a lamp (start)
            elif line[i][0] == "Q":

current_line_construction.append(LampInput(int(line[i][1:])))
            # if position is a white sensor
            elif line[i] == "W":
                current_line_construction.append(White())

current_line_construction.append(current_line_construction[-1])
                i += 1
            # if position is a blue sensor
            elif line[i] == "B":
                current_line_construction.append(Blue())

current_line_construction.append(current_line_construction[-1])
                i += 1
            # if position is a lamp (end)
            elif line[i][0] == "L":

current_line_construction.append(LampOutput(int(line[i][1:])))
            elif i+1 < self.x:
                # if position is a red sensor with sensor position 0
                if line[i] == "R" and line[i+1] == "r":
                    current_line_construction.append(Red(0))

current_line_construction.append(current_line_construction[-1])
                    i += 1
                # if position is a red sensor with sensor position 1
                elif line[i] == "r" and line[i+1] == "R":
                    current_line_construction.append(Red(1))

current_line_construction.append(current_line_construction[-1])
                    i += 1

            # go to next position
            i += 1
        # add line to construction
        self.construction.append(current_line_construction)
```

[C1] Algorithmus zur Erstellung der Konstruktion

```python
def get_combinations_of_inputs(self):
    combinations = []
    for i in range(2**len(self.inputs)):
        c_combination = list(bin(i)[2:].zfill(len(self.inputs)))
        c_combination = [int(x) for x in c_combination]
        combinations.append(c_combination)
    return combinations
```

[C2] Kombinationsgenerator

```python
def get_res(self, combination):
    # set inputs to combination
    for i in range(len(self.inputs)):
        self.inputs[i].state = combination[i]

    # calculate outputs
    # for each line in the construction (y-value)
    for y in range(self.y):
        # for each position in line (x-value)
        for x in range(self.x):
            # if brick is None (empty)
            if self.construction[y][x] is None:
                pass

            # if brick is a lamp input
            elif self.construction[y][x].type == "lamp_in":
                pass

            # if brick is a red sensor
            elif self.construction[y][x].type == "red":
                # get position of the sensor in the red sensor block
                sensor_position = self.construction[y][x].sensor_position
                # get current side of the red sensor
                current_side = self.get_c_side_of_sensor(x, y)
                # if sensor of the sensor block is on the current side
                if sensor_position == current_side:
                    # set output of the sensor

self.construction[y][x].input(self.get_output_of_brick_in_line_above(x, y))
                else:
                    pass

            # if brick is a white sensor
            elif self.construction[y][x].type == "white":
                current_side = self.get_c_side_of_sensor(x, y)
                # if c_pos is on the left
                if current_side == 0:

self.construction[y][x].input(self.get_output_of_brick_in_line_above(x, y),

self.get_output_of_brick_in_line_above(x+1, y))
```

```
                # if c_pos is on the right
                elif current_side == 1:

self.construction[y][x].input(self.get_output_of_brick_in_line_above(x-1,
y),

self.get_output_of_brick_in_line_above(x, y))
            # if brick is a blue sensor
            elif self.construction[y][x].type == "blue":
                current_side = self.get_c_side_of_sensor(x, y)
                # if c_pos is on the left
                if current_side == 0:

self.construction[y][x].input(self.get_output_of_brick_in_line_above(x, y),

self.get_output_of_brick_in_line_above(x+1, y))
                # if c_pos is on the right
                elif current_side == 1:

self.construction[y][x].input(self.get_output_of_brick_in_line_above(x-1,
y),

self.get_output_of_brick_in_line_above(x, y))
            # if brick is a lamp output
            elif self.construction[y][x].type == "lamp_out":
                self.construction[y][x].state =
self.get_output_of_brick_in_line_above(x, y)

    # get outputs of the construction (lamp outputs)
    outputs = []
    for output in self.outputs:
        outputs.append(output.state)
    return outputs
```

[C3] Algorithmus zur Ermittlung der Ausgaben für eine bestimmte Kombinationen an Eingaben

```
def save_results(self):
    # text file with table
    # create table with pandas
    data = {}
    # get values of input i and add input i to data
    for i in range(len(self.inputs)):
        c_q = "Q" + str(self.inputs[i].number)
        c_ins = []
        for j in range(len(self.combinations_of_inputs)):
            c_ins.append(self.combinations_of_inputs[j][i])
        data[c_q] = c_ins
    # get values of output i and add output i to data
    for i in range(len(self.outputs)):
        c_l = "L" + str(self.outputs[i].number)
        c_outs = []
```

```
        for j in range(len(self.results)):
            c_outs.append(self.results[j][i])
        data[c_l] = c_outs
    # create table
    # print("data:", data)
    df = pd.DataFrame(data)
    # save dataframe to txt file without index in line
    f = open("result_a4_" + self.filename + "_table.txt", "w")
    f.write(df.to_string(index=False))
    f.close()
```

[C4] Programm zur Speicherung der ermittelten Ergebnisse

```
def get_output_of_brick_in_line_above(self, x, y):
    # if brick is in first line of construction
    if y == 0:
        return None
    # if brick is not in first line of construction
    else:
        # if brick above is a red sensor
        if self.construction[y-1][x].type == "red":
            return self.construction[y-1][x].output_0

        # if brick above is a white sensor
        elif self.construction[y-1][x].type == "white":
            return self.construction[y-1][x].output_0

        # if brick above is a blue sensor
        elif self.construction[y-1][x].type == "blue":
            # if c_pos is on the left side of the current sensor
            if self.get_c_side_of_sensor(x, y) == 0:
                return self.construction[y-1][x].output_0
            # if c_pos is on the right of the current sensor
            elif self.get_c_side_of_sensor(x, y) == 1:
                return self.construction[y-1][x].output_1
            # if c_pos is on the lamp output of the blue brick above
            elif self.get_c_side_of_sensor(x, y) == 2:
                # if left side of blue brick above is over lamp output
                if self.get_c_side_of_sensor(x, y-1) == 0:
                    return self.construction[y-1][x].output_0
                # if right side of blue brick above is over lamp output
                elif self.get_c_side_of_sensor(x, y-1) == 1:
                    return self.construction[y-1][x].output_1

        # if brick above is a lamp input
        elif self.construction[y-1][x].type == "lamp_in":
            return self.construction[y-1][x].state

        # if brick above is none (empty)
        elif self.construction[y-1][x] is None:
            return 0
```

[C5] Programm um die Ausgabe des Bausteins über dem aktuellen Baustein zu ermitteln