Aufgabe 4: „Nandu“

Team-ID: 00128

Team-Name: E29C8CF09F8E89

Bearbeiter/-innen dieser Aufgabe:
Leonhard Masche

20. Oktober 2023

Inhaltsverzeichnis

1 Lösungsidee

Die Bausteine aus der Aufgabe können leicht als Lambda-Funktionen modelliert werden. Die Aufgabe wird als Matrix geladen und es wird über die einzelnen Zeilen und Buchstaben iteriert um Bausteine zu finden. Licht-Zustände werden in einer eigenen

Matrix gespeichert, in der zu Anfang die Eingabezustände eingetragen werden. Wenn beim Iterieren zwei Buchstaben gefunden werden, die zu einem bekannten Baustein passen, so werden die Eingabezustände des Bausteins aus der Licht-Zustands-Matrix geladen und die Ausgabe mithilfe der Lambda-Funktion des Baustein errechnet, welche dann wiederum in die Licht-Zustands-Matrix eingefügt wird. Nachdem ein Baustein bearbeitet wurde, wird (wenn möglich) gleich zwei Felder nach rechts gesprungen, um ein wiederholtes Anwenden eines Bausteins zu vermeiden. Dies wird über alle Zeilen der Konstruktion (bis auf die Letzte mit ausschließlich Ausgabe-Lampen) fortgeführt. Zuletzt wird das Ergebnis an den Ausgabe-Lampen ausgelesen. Um alle möglichen Eingaben für ein Konstrukt zu simulieren gibt es bei n Eingabe-Lampen n^2 Möglichkeiten für unterschiedliche Ausgaben, welche alle durch den vorher genannten Prozess simuliert und in einer Tabelle notiert werden.

2 Umsetzung

Das Programm (`program.py`) ist in Python umgesetzt und mit einer Umgebung ab der Version 3.6 ausführbar. Zum Umgang mit Matrizen wird die externe Bibliothek `numpy`, für die Verwendung von Tabellen `pandas` verwendet. Alle Vorraussetzungen für das Ausführen des Programmes können mit dem Befehl `pip install -r requirements.txt` installiert werden.

Beim Ausführen der Datei wird zuerst nach der Zahl des Beispiels gefragt. Dieses wird nun aus der Datei `input/zauberschule{n}.txt` geladen und bearbeitet. Nun werden durch einen einfachen binären Zähler alle Eingabezustände simuliert und das Ergebnis ausgegeben. Zusätzlich wird es auch als `csv`-Datei im Ordner `output` gespeichert, was eine programmatische Verifizierung der Ergebnisse erleichtert.

3 Beispiele

Hier wird das Programm auf die 5 Beispiele von der Website, sowie das linke Beispiel aus der Aufgabenstellung (`nandu0.txt`) angewendet:

- **nandu0.txt**

  ```
  Simuliert in 0.80ms
  ```

```
    Q1   Q2   L1   L2
0  Aus  Aus  Aus  Aus
1  Aus   An  Aus  Aus
2   An  Aus  Aus  Aus
3   An   An   An   An

Ausgabe gespeichert in "output/nandu0.csv"
```

- **nandu1.txt**

```
Simuliert in 0.80ms

    Q1   Q2   L1   L2
0  Aus  Aus   An   An
1  Aus   An   An   An
2   An  Aus   An   An
3   An   An  Aus  Aus

Ausgabe gespeichert in "output/nandu1.csv"
```

- **nandu2.txt**

```
Simuliert in 1.09ms

    Q1   Q2   L1   L2
0  Aus  Aus  Aus   An
1  Aus   An  Aus   An
2   An  Aus  Aus   An
3   An   An   An  Aus

Ausgabe gespeichert in "output/nandu2.csv"
```

- **nandu3.txt**

```
Simuliert in 1.60ms

    Q1   Q2   Q3   L1   L2   L3   L4
0  Aus  Aus  Aus   An  Aus  Aus   An
1  Aus  Aus   An   An  Aus  Aus  Aus
```

```
2  Aus   An  Aus   An  Aus   An   An
3  Aus   An   An   An  Aus   An  Aus
4   An  Aus  Aus  Aus   An  Aus   An
5   An  Aus   An  Aus   An  Aus  Aus
6   An   An  Aus  Aus   An   An   An
7   An   An   An  Aus   An   An  Aus

Ausgabe gespeichert in "output/nandu3.csv"
```

- **nandu4.txt**

```
Simuliert in 1.95ms

     Q1   Q2   Q3   Q4   L1   L2
0   Aus  Aus  Aus  Aus  Aus  Aus
1   Aus  Aus  Aus   An  Aus  Aus
2   Aus  Aus   An  Aus  Aus   An
3   Aus  Aus   An   An  Aus  Aus
4   Aus   An  Aus  Aus   An  Aus
5   Aus   An  Aus   An   An  Aus
6   Aus   An   An  Aus   An   An
7   Aus   An   An   An   An  Aus
8    An  Aus  Aus  Aus  Aus  Aus
9    An  Aus  Aus   An  Aus  Aus
10   An  Aus   An  Aus  Aus   An
11   An  Aus   An   An  Aus  Aus
12   An   An  Aus  Aus  Aus  Aus
13   An   An  Aus   An  Aus  Aus
14   An   An   An  Aus  Aus   An
15   An   An   An   An  Aus  Aus

Ausgabe gespeichert in "output/nandu4.csv"
```

- **nandu5.txt**

```
Simuliert in 15.92ms

     Q1   Q2   Q3   Q4   Q5   Q6   L1   L2   L3   L4   L5
0   Aus  Aus  Aus  Aus  Aus  Aus  Aus  Aus  Aus   An  Aus
1   Aus  Aus  Aus  Aus  Aus   An  Aus  Aus  Aus   An  Aus
2   Aus  Aus  Aus  Aus   An  Aus  Aus  Aus  Aus   An   An
3   Aus  Aus  Aus  Aus   An   An  Aus  Aus  Aus   An   An
4   Aus  Aus  Aus   An  Aus  Aus  Aus  Aus   An  Aus  Aus
..  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...
59   An   An   An  Aus   An   An   An  Aus  Aus   An   An
60   An   An   An   An  Aus  Aus   An  Aus   An  Aus  Aus
61   An   An   An   An  Aus   An   An  Aus   An  Aus  Aus
62   An   An   An   An   An  Aus   An  Aus  Aus   An   An
63   An   An   An   An   An   An   An  Aus  Aus   An   An

[64 rows x 11 columns]

Ausgabe gespeichert in "output/nandu5.csv"
```

output/nandu5.csv: https://github.com/leonhma/bwinf-42-1/blob/main/a4-nandu/output/nandu5.csv

4 Quellcode

program.py

```python
import os
import time
from itertools import product
from typing import Dict, List, Tuple

import numpy as np
import pandas as pd


def load_board(
    path: str,
```

```
) -> Tuple[int, int, np.ndarray, List[Tuple[str, int]], List[Tuple[str, int]]]:
    """
    Öffne ein Beispiel und gebe die Konstruktion zurück.

    Parameters
    ----------
    path : str
        Der Dateipfad der Beispieldatei relativ zur `program.py`-Datei.

    Returns
    -------
    Tuple[int, int, np.ndarray, List[Tuple[str, int]], List[Tuple[str, int]]]
        - n, m (Größen der Konstruktion)
        - Ein 2D-Array mit den Zeichen der Konstruktion.
        - Liste der Input-Positionen (Lampen)
            - Name der Lampe
            - x-Position
        - Liste der Output-Positionen (Lampen)
            - Name der Lampe
            - x-Position
    """
    with open(os.path.join(os.path.dirname(__file__), path), "r", encoding="utf8") as
    ↪  f:
        # Dimensionen der Konstruktion
        n, m = map(int, f.readline().split())

        board = np.empty((n, m), dtype=str)
        inp = []
        out = []

        for mi in range(m):
            for ni, c in enumerate(f.readline().split()[:n]):
                board[ni][mi] = c  # Charakter in `board` speichern
                if c.startswith("Q"):
                    inp.append((c, ni))  # mi ist 0, muss nicht gespeichert werden
                elif c.startswith("L"):
                    out.append((c, ni))  # mi ist m-1, muss nicht gespeichert werden

        assert len(inp) > 0, "Keine Lampe als Eingabe konfiguriert!"
        assert len(out) > 0, "Keine Lampe als Ausgabe konfiguriert!"
        return n, m, board, inp, out

```

```python

# Bausteine werden durch lambda-Funktionen modelliert
kernels = {
    "rR": lambda a, b: (not b, not b),  # beide wenn Eingabe bei R 0 ist
    "Rr": lambda a, b: (not a, not a),  # beide wenn Eingabe bei R 0 ist
    "WW": lambda a, b: (not (a and b), not (a and b)),  # beide, solange nicht beide
    ↪ Eingaben 1 sind
    "BB": lambda a, b: (a, b),  # Eingabe wird weitergeleitet
}


def simulate(
    n: int,
    m: int,
    board: np.ndarray,
    inp: List[Tuple[str, int]],
    out: List[Tuple[str, int]],
    inp_states: tuple[bool, ...],
) -> Dict[str, bool]:
    """
    Simuliere die Konstruktion.

    Parameters
    ----------
    n : int
        Breite der Konstruktion.
    m : int
        Höhe der Konstruktion (Anzahl der Zeilen).
    board : np.ndarray
        2D-Array mit den Zeichen der Konstruktion.
    inp : List[Tuple[str, int]]
        Liste der Input-Positionen (Lampen)
            - Name der Lampe
            - x-Position
    out : List[Tuple[str, int]]
        Liste der Output-Positionen (Lampen)
            - Name der Lampe
            - x-Position
    inp_states : tuple[bool, ...]
        Eingabezustände.

    Returns
```

```
94      -------
95      Dict[str, bool]
96          Ausgabezustände.
97      """
98      # Matrix für die Lichtzustände an einzelnen Positionen
99      states = np.zeros((n, m), dtype=bool)
100     # Eingabezustände setzen
101     for inp, state in zip(inp, inp_states):
102         states[inp[1]][0] = state
103         # print(f'states[{inp[1]}][0] auf {state} gesetzt. ("{inp[0]}")')
104
105     # Simulieren
106     for mi in range(1, m - 1):  # für jede Zeile...
107         ni = 0
108         while True:  # ...wird von links durch die Positionen iteriert
109             # überprüfen, ob es sich bei den nächsten beiden Zeichen um einen
                ↪ Baustein handelt
110             kname = board[ni][mi] + board[ni + 1][mi]
111             kernel = kernels.get(kname, None)
112             if kernel:
113                 # Eingabezustände werden aus der Matrix gelesen
114                 a = states[ni][mi - 1]
115                 b = states[ni + 1][mi - 1]
116                 # Ausgabe des einzelnen Baustein wird berechnet
117                 c, d = kernel(a, b)
118                 # Ausgabezustände setzen
119                 states[ni][mi] = c
120                 states[ni + 1][mi] = d
121                 # die nächste Position wird übersprungen, da sie noch zum jetzigen
                    ↪ Baustein gehört
122                 ni += 1
123                 # print(f"states[{ni}][{mi}] auf {c} und states[{ni+1}][{mi}] auf {d}
                    ↪ gesetzt.")
124             # else:
125             #   print(f'kein kernel gefunden für "{kname}"')
126             ni += 1
127             if ni >= n - 1:
128                 break
129     # Ausgabezustände auslesen und zurückgeben
130     return {out[0]: states[out[1]][m - 2] for out in out}
131
132
```

```python
def main(
    n: int,
    m: int,
    board: np.ndarray,
    inp: List[Tuple[str, int]],
    out: List[Tuple[str, int]],
    n_bsp: int,
):
    t_start = time.time()  # zeitmessung starten
    results = []
    for inp_states in product(
        [False, True], repeat=len(inp)
    ):  # für jede mögliche Kombination der Eingabe wird simuliert
        out_states = simulate(n, m, board, inp, out, inp_states)
        results.append(
            {
                **{inp[0]: inp_state for inp, inp_state in zip(inp, inp_states)},
                **out_states,
            }
        )

    results = pd.DataFrame(results)
    results = results.map(lambda x: "An" if x else "Aus")
    print(f"Simuliert in {((time.time() - t_start)*1000):.2f}ms")
    print()
    print(results)
    print()
    results.to_csv(f"output/nandu{n_bsp}.csv", index=False)
    print(f'Ausgabe gespeichert in "output/nandu{n_bsp}.csv"')


# Haupt-Loop
while True:
    try:
        n_bsp = int(input("Bitte Nummer des Beispiels eingeben:\n> "))
        n, m, board, inp, out = load_board(f"input/nandu{n_bsp}.txt")
        print()
        main(n, m, board, inp, out, n_bsp)
    except Exception as e:  # Error-Handling
        print(f"{e.__class__.__name__}: {e}")
    print()
```