Aufgabe 4: Nandu

Team-ID: 00779
Team: tiantianquan
Autor: Florian Werth
18. November 2023

Inhaltsverzeichnis

Lösungsidee

Um die Eingabe zu verarbeiten wird jede Reihe als String mit Leerzeichen, Q's, und L's entfernt in einem Feld ***Bausteine*** gespeichert, damit man damit gut arbeiten kann.

Zuerst: Wie kann man bei gegebenen Zustand die Ausgabe vorhersagen?

In einem 2D-Array ***Zustand*** speichert man, ob eine Stelle aktiv ist. In der ersten Zeile setzt man jede Position auf aktiv, bei der auch ein Eingang aktiv ist.

Danach geht man alle anderen Reihen von links nach rechts durch, stößt man auf einen Sensor, dann

Bausteine[X][Y] == 'W': ist Zustand bei [X][Y+1] und [X+1][Y+1] == 'aktiv',

dann Zustand bei [X][Y] und [X+1][Y] = 'aus'

Ansonsten Zustand bei [X][Y] und [X+1][Y] = 'aktiv'

X += 2

Bausteine[X][Y] == bei 'r': ist Zustand bei [X+1][Y+1] == 'aktiv'

dann Zustand bei [X][Y] und [X+1][Y] = 'aus'

Ansonsten Zustand bei [X][Y] und [X+1][Y] = 'aktiv'

X += 2

Bausteine[X][Y] == bei 'R' ist Zustand bei [X][Y+1] == 'aktiv'

dann Zustand bei [X][Y] und [X+1][Y] = 'aus'

Ansonsten Zustand bei [X][Y] und [X+1][Y] = 'aktiv'

X += 2

Bausteine[X][Y] == bei 'B' ist Zustand bei [X][Y+1] == 'aktiv'

dann Zustand bei [X][Y] = 'aktiv'

Ansonsten Zustand bei [X][Y] = 'aus'

X += 1

bei 'X':
X += 1

Zum Schluss ist der Zustand der Ausgabelampen gleich dem Zustand über der Lampe.
Das wiederholt man für alle möglichen Werte der Eingänge.

Umsetzung

Die Lösung wurde in C++ geschrieben.

Bausteine einlesen

Die ***Bausteine*** werden in einem Vektor `std::vector<std::string>` *bausteine* gespeichert. Jede Zeile wird in einer for-Schleife eingelesen. Leerzeichen, Qs und Ls werden aus der Zeile gelöscht:

```
line.erase(remove(line.begin(), line.end(), ' '), line.end());
line.erase(remove(line.begin(), line.end(), 'Q'), line.end());
line.erase(remove(line.begin(), line.end(), 'L'), line.end());
```

Danach wird die Zeile dem Vektor hinten angehangen.

```
bausteine.push_back(line);
```

Ausgangszustände berechnen

Eine Funktion **calculateOutput** berechnet für gegebene Eingangswerte die Ausgangswerte. Das Vorgehen ist dabei wie in der Lösungsidee beschrieben.

Es wird der Vektor

```
std::vector<std::vector<bool>> zustand(height, std::vector<bool>(width));
```

definiert.

In den Zustandsvektor werden die Eingangswerte geschrieben.

```
for (auto& input : inputs)
	zustand[0][input.pos] = input.active;
```

In einer doppelten for-Schleife wird von der ersten Zeile nach den Eingängen bis zur vorletzten und von links nach rechts, der zustand-Vektor gefüllt, so wie in der Lösungsidee beschrieben.

Nachdem alle Zustände berechnet wurden, werden die Werte der Ausgänge gesetzt.

```
for (auto& output : outputs)
{
	output.active = zustand[height - 2][output.pos];
}
```

die Funktion muss für jeden möglichen Eingangszustand aufgerufen werden.

Eingangszustände generieren

Eine kurze rekursive Funktion generiert alle möglichen Eingangswerte. Es gibt ein Feld, das die derzeitigen Eingangswerte enthält. Am Anfang sind alle Eingangswerte auf Null gesetzt. Für jedes Element wird die Funktion zweimal neu aufgerufen bis alle Elemente durchlaufen sind. Dann wird das Feld mit den derzeitigen Werten in einen Vektor mit allen möglichen Werten gespeichert.

```
void generateInputStates(int amount, std::vector<std::vector<bool>>& allStates,
std::vector<bool> currentState, int i)
{
	i++;
	if (i == amount)
	{
		states.push_back(currentState);
		return;
	}
	generateInputStates(amount, allStatestates, currentState, i);
	currentState[i] = true;
	generateInputStates(amount, allStates, currentState, i);
}
```

Beispiele

nandu1.txt

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

der 2D-Zustandsvektor von Q1=0, Q2=0

```
   Q1Q2
0 | 0 | 0 | 0 |
0 | 1 | 1 | 0 | W ist aktiviert, da darüber nicht zwei aktive Felder sind
0 | 0 | 0 | 0 | rR und Rr sind aus, da über den Rs aktive Felder ist
0 | 0 | 0 | 0 | B leitet nur das vorherige Signal weiter
0 | 1 | 1 | 0 | W ist aktiv da, darüber nicht zwei aktive Felder sind
0 | 0 | 0 | 0 | Die Felder über L1 und L2 sind aktiv, daher sind L1 und L2 an
   L1  L2
```

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	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

nandu4.txt

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

nandu5.txt

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

0	1	1	1	1	1		0	0	0	1	1
1	0	0	0	0	0		1	0	0	1	0
1	0	0	0	0	1		1	0	0	1	0
1	0	0	0	1	0		1	0	0	1	1
1	0	0	0	1	1		1	0	0	1	1
1	0	0	1	0	0		1	0	1	0	0
1	0	0	1	0	1		1	0	1	0	0
1	0	0	1	1	0		1	0	0	1	1
1	0	0	1	1	1		1	0	0	1	1
1	0	1	0	0	0		1	0	0	1	0
1	0	1	0	0	1		1	0	0	1	0
1	0	1	0	1	0		1	0	0	1	1
1	0	1	0	1	1		1	0	0	1	1
1	0	1	1	0	0		1	0	1	0	0
1	0	1	1	0	1		1	0	1	0	0
1	0	1	1	1	0		1	0	0	1	1
1	0	1	1	1	1		1	0	0	1	1
1	1	0	0	0	0		1	0	0	1	0
1	1	0	0	0	1		1	0	0	1	0
1	1	0	0	1	0		1	0	0	1	1
1	1	0	0	1	1		1	0	0	1	1
1	1	0	1	0	0		1	0	1	0	0
1	1	0	1	0	1		1	0	1	0	0
1	1	0	1	1	0		1	0	0	1	1
1	1	0	1	1	1		1	0	0	1	1
1	1	1	0	0	0		1	0	0	1	0
1	1	1	0	0	1		1	0	0	1	0
1	1	1	0	1	0		1	0	0	1	1
1	1	1	0	1	1		1	0	0	1	1
1	1	1	1	0	0		1	0	1	0	0
1	1	1	1	0	1		1	0	1	0	0
1	1	1	1	1	0		1	0	0	1	1
1	1	1	1	1	1		1	0	0	1	1

Quellcode

Ausgang berechnen

```
void calculateOutput(std::vector<std::string> inputMatrix, const std::vector<Field>&
inputs, std::vector<Field>& outputs, int height, int width)
{
	std::vector<std::vector<bool>> zustand(height, std::vector<bool>(width));

	for (auto& input : inputs)
	{
		zustand[0][input.pos] = input.active;
	}

	for (int y = 1; y < height - 1; y++)
	{
		for (int x = 0; x < width; x++)
		{
			switch (inputMatrix[y][x])
			{
			case 'W':
				if (!(zustand[y - 1][x] && zustand[y - 1][x + 1]))
				{
					zustand[y][x] = true;
					zustand[y][x + 1] = true;
				}
				x += 1;
				break;

			case 'r':
				if (!zustand[y - 1][x + 1])
				{
					zustand[y][x] = true;
					zustand[y][x + 1] = true;
				}
				x += 1;
				break;

			case 'R':
				if (!zustand[y - 1][x])
				{
					zustand[y][x] = true;
					zustand[y][x + 1] = true;
				}
				x += 1;
				break;

			case 'B':
				if (zustand[y - 1][x])
				{
					zustand[y][x] = true;
				}
			}
		}
	}
	for (auto& output : outputs)
	{
		output.active = zustand[height - 2][output.pos];
	}
```

```
}
```

Konstruktion einlesen

```
std::vector<std::string> readInput(std::ifstream& file, int height)
{
	std::vector<std::string> inputMatrix;
	for (int i = 0; i < height; i++)
	{
		std::string line;
		std::getline(file, line);
		line.erase(remove(line.begin(), line.end(), ' '), line.end());
		line.erase(remove(line.begin(), line.end(), 'Q'), line.end());
		line.erase(remove(line.begin(), line.end(), 'L'), line.end());
		inputMatrix.push_back(line);
	}
	return inputMatrix;
}
```

Eingänge und Ausgänge finden

```
for (int x = 0; x < width; x++)
{
	if (inputMatrix[0][x] != 'X')
	{
		inputs.push_back({ x, false });
	}
	if (inputMatrix[height-1][x] != 'X')
	{
		outputs.push_back({ x, false });
	}
}
```

Eingangszustände generieren

```
void generateInputStates(int amount, std::vector<std::vector<bool>>& states,
std::vector<bool> currentState, int i)
{
	i++;
	if (i == amount)
	{
		states.push_back(currentState);
		return;
	}
	generateInputStates(amount, states, currentState, i);
	currentState[i] = true;
	generateInputStates(amount, states, currentState, i);
}
```

Alle Ausgänge berechnen und ausgeben

```
for (auto inputState : allInputStates)
{
	for (int x = 0; x < inputs.size(); x++)
	{
		inputs[x].active = inputState[x];
	}
	calculateOutput(inputMatrix, inputs, outputs, height, width);
	for (auto& input : inputs)
	{
		std::cout << input.active << "    ";
	}
	for (auto& output : outputs)
	{
		std::cout << output.active << "    ";
		output.active = false;
	}

	std::cout << "\n";
}
```