1. Runde

des

42. Bundeswettbewerbs Informatik

# Aufgabe 4: Nandu

# Teamname: EZ

Team-ID: 00871

Bearbeitet

von

Florian Bange
Teilnahme-ID: 71639

21. November 2023

# Inhaltsverzeichnis

# 0. Einleitung

Dieses Dokument beinhaltet meine Dokumentation der vierte Aufgabe[1] der ersten Runde des 42. Bundeswettbewerbs Informatik[2] aus dem Jahr 2023.

Fuer diese Aufgabe wird zunaechst das gegebene Problem mathematisch definiert. Darauf folgt ein Loesungsvorschlag, welcher anschlieszend bezueglich seiner Zeit- und Platzkomplexitaet analysiert wird.
Weiter werden Details zur Implementierung in C++ gegeben, woraufhin die Beispiele der BwInf Website und der Quellcode folgen.

# 1. Modellierung und Definition des Problems

Im Folgenden ist mit Eingabe die Darstellung einer Konstruktion ohne die Anzahl der Zeilen und Spalten ebendieser.
Weiter sei im Folgenden $n \in \mathbb{N}$ die Anzahl der Zeilen der Konstruktion der Eingabe und $k \in \mathbb{N}$ die Anzahl der Spalten ebendieser. Dann besteht die Konstruktion in der Eingabe aus $n$ Zeilen mit je $k$ durch Leerzeichen getrennte Zeichenketten. Im Folgenden wird eine Konstruktion gemaesz der Aufgabenstellung mathematisch definiert.

### Definition 1: Konstruktion

Eine Konstruktion im Sinne der Aufgabenstellung besteht aus einer Matrix $X$ mit $n \in \mathbb{N}$ Zeilen und $k \in \mathbb{N}$ Spalten, sodass $n \geq 2$ und $k \geq 1$.
Dabei beinhaltet die erste Zeile eine Anzahl $q \in \mathbb{N}$ mit $0 \leq q \leq k$ an Quellen $Q1,\ Q2,\ \dots\ Qq$ an den Positionen

$$Q = (Q_1,\ Q_2,\ \dots\ Q_q)$$

mit $1 \leq Q_1 < Q_2 < \dots < Q_q \leq k$ und die letzte Zeile beinhaltet $l \in \mathbb{N}$ mit $0 \leq l \leq k$ LEDs $L1,\ L2,\ \dots\ ,\ Ll$ an den Positionen

$$L = (L_1,\ L_2,\ \dots\ ,\ L_l)$$

mit $1 \leq L_1 < L_2 < \dots < L_l \leq k$. Die restlichen Eintraege sind $X$.
Die mittleren $n-2$ Zeilen enthalten Bausteine, die durch die Zeichen $W$, $B$, $R$ und $r$ dargestellt werden. Dabei besteht ein Baustein stets aus zwei Zeichen, die in einer der $n-2$ Zeilen je direkt nebeneinander in den Spalten $i$ und $i+1$ sind, wobei $i \in \{1,\ 2, \dots\ ,\ k-1\}$. Die restlichen Zeichen sind erneut $X$.
Es werden vier Typen von Bausteinen unterschieden:

1. Typ: $W$ gefolgt von $W$ - $(W,\ W)$,
2. Typ: $B$ gefolgt von $B$ - $(B,\ B)$,
3. Typ: $R$ gefolgt von $r$ - $(R,\ r)$, und
4. Typ: $r$ gefolgt von $R$ - $(r,\ R)$.

### Beispiel 2:

Wir betrachten die folgende Konstruktion:

$$X = \left(\begin{array}{cccc} X & Q1 & Q2 & X \\ \hline X & W & W & X \\ r & R & R & r \\ X & B & B & X \\ X & W & W & X \\ \hline X & L1 & L2 & X \end{array}\right)$$

Dann ist $n = 6$ und $k = 4$. Weiter beinhaltet die erste Zeile $q = 2$ Lichtquellen $Q1$ und $Q2$ mit den Positionen $Q = (2,\ 3)$ und die letzte Zeile $l = 2$ LEDs $L1$ und $L2$ and den Positionen $L = (2,\ 3)$.
Die Bausteine der mittleren $n - 2 = 4$ Zeilen sind die Folgenden:

[1] siehe `https://bwinf.de/fileadmin/bundeswettbewerb/42/BwInf_42_Aufgaben_WEB.pdf`
[2] siehe `https://bwinf.de/bundeswettbewerb/42/1/`

1. Typ 1: $(W,\ W)$ in Zeile 2 und den Spalten 2 und 3, sowie in Zeile 5 und den Spalten 2 und 3,
2. Typ 2: $(B,\ B)$ in Zeile 4 und den Spalten 2 und 3,
3. Typ 3: $(R,\ r)$ in Zeile 3 und den Spalten 3 und 4, und
4. Typ 4: $(r,\ R)$ in Zeile 3 und den Spalten 1 und 2.

## Definition 3: Funktionsweise der Bausteine

Jeder der vier zuvor definierten Bausteine reagiert auf binaere Eingaben an der ersten und zweiten Position des Tupels und gibt entspraechend der folgenden Definitionen zwei binaere Ausgaben.
In den folgenden Definitionen der Bausteine wird jeweils eine Tabelle angegeben, in welcher in jeder Spalte in der ersten Zeile die Eingabe und in der zweiten Zeile die dazugehoerige Ausgabe angegeben ist.

1. Typ 1: $(W,\ W)$.

| $(0,\ 0)$ | $(0,\ 1)$ | $(1,\ 0)$ | $(1,\ 1)$ |
|---|---|---|---|
| $(1,\ 1)$ | $(1,\ 1)$ | $(1,\ 1)$ | $(0,\ 0)$ |

2. Typ 2: $(B,\ B)$.

| $(0,\ 0)$ | $(0,\ 1)$ | $(1,\ 0)$ | $(1,\ 1)$ |
|---|---|---|---|
| $(0,\ 0)$ | $(0,\ 1)$ | $(1,\ 0)$ | $(1,\ 1)$ |

3. Typ 3: $(R,\ r)$.

| $(0,\ 0)$ | $(0,\ 1)$ | $(1,\ 0)$ | $(1,\ 1)$ |
|---|---|---|---|
| $(1,\ 1)$ | $(1,\ 1)$ | $(0,\ 0)$ | $(0,\ 0)$ |

4. Typ 4: $(r,\ R)$.

| $(0,\ 0)$ | $(0,\ 1)$ | $(1,\ 0)$ | $(1,\ 1)$ |
|---|---|---|---|
| $(1,\ 1)$ | $(0,\ 0)$ | $(1,\ 1)$ | $(0,\ 0)$ |

Dabei wurde auch fuer die letztes beiden Typen ein Paar als Eingabe verwendet, um den unterschied zwischen ersten und zweiten Komponent treffen zu koennen.

## Bemerkung 4: Alternative Definitionen der Funktionsweisen der Bausteine

Fuer jeden der soeben definierten Bausteine koennen mit Hilfe von logischen Operatoren alternative, vereinfachte Definitionen gegeben werden. Dazu sei je $(a,\ b)$ die Eingabe in den Baustein mit $a,\ b \in \{0,\ 1\}$

1. Typ 1: $(W,\ W)$.
   Hier sind stets beide Komponenten der Ausgabe null genau dann, wenn $a = 1 = b$. Ansonsten sind beide eins. Daraus ergibt sich fuer beide Komponenten die Formel $\neg(a \wedge b)$. Somit ist die Ausgabe

   $$(\neg(a \wedge b),\ \neg(a \wedge b)).$$

2. Typ 2: $(B,\ B)$.
   Hier entspricht die Ausgabe stets der Eingabe. Somit ergibt sich die Ausgabe $(a,\ b)$.

3. Typ 3: $(R,\ r)$.
   Hier entspechen beide Komponenten der Ausgabe eins genau dann, wenn die erste Komponente der Eingabe null ist, und sonst null. Somit ergibt sich die Ausgabe

   $$(\neg a,\ \neg a)$$

4. Typ 4: $(r,\ R)$.
   Analog zum vorherigen Typen, entsprechen beide Komponenten der Ausgabe eins genau dann, wenn die zweite Komponente der Eingabe null ist, und sonst null. Somit ergibt sich die Ausgabe

   $$(\neg b,\ \neg b)$$

### Definition 5: Reaktion einer Zeile mit Bausteinen auf eine Eingabe

Es sei $X$ eine Konstruktion mit $n$ Zeilen und $k$ Spalten. Nun wird eine Zeile $i$ mit $i \in \{2,\ 3,\ \dots\ ,\ n-1\}$ betrachet. Fuer eine solche Zeile sei eine Eingabe definiert als ein $k$-Tupel

$$x \in \{0,\ 1\}^k,$$

sodass

$$x = (x_1,\ x_2,\ \dots\ ,\ x_k)$$

mit $x_j \in \{0,\ 1\}$ fuer alle $j \in \{1,\ 2,\ \dots\ ,\ k\}$.
Dann ist die Reaktion oder Ausgabe der Zeile $i$ fuer die Eingabe $x$ gegeben durch ein $k$-Tupel

$$t = (t_1,\ t_2,\ \dots\ ,\ t_k)$$

wobei fuer $j \in \{1,\ 2,\ \dots\ ,\ k\}$ Folgendes gilt:

1. Falls der Eintrag der Konstruktion $X$ in der Zeile $i$ und Spalte $j$ $X$ ist, ist $t_i = 0$.
2. Falls die Eintraege der Konstruktion $X$ in der Zeile $i$ und den Spalten $j$ und $j+1$ einen Baustein bilden, ist $(t_j,\ t_{j+1})$ gleich der Ausgabe des Bausteins fuer die Eingabe $(x_j,\ x_{j+1})$.

### Definition 6: Zustaende der Lichtquellen

Es sei $X$ eine Konstruktion mit $n$ Zeilen und $k$ Spalten. Dann befinden sich in der ersten Zeile der Matrix $X$ $q$ Lichtquellen an den Positionen

$$Q = (Q_1,\ Q_2,\ \dots\ ,\ Q_q).$$

Nun nennen wir ein $q$-Tupel

$$\vec{Q} \in \{0,\ 1\}^q,$$

sodass

$$\vec{Q} = (x_1,\ x_2,\ \dots\ ,\ x_q)$$

mit $x_i \in \{0,\ 1\}$ fuer alle $i \in \{1,\ 2,\ \dots\ ,\ q\}$, einen Zustand der Lichtquellen.
Weiter ist fuer einen Zustand $x$ der Lichtquellen $x_i$ der Zustand der $i$-ten Lichtquelle fuer $i \in \{1,\ 2,\ \dots\ ,\ q\}$.

### Bemerkung 7:

Da ein Zustand der Lichtquellen einer Konstruktion $X$ eine $q$-Variation aus einer 2-Menge mit Wiederholungen ist, ergeben sich genau $2^q$ moegliche Zustaende der Lichtquellen.

### Definition 8: Reaktion der LEDs auf einen Zustand der Lichtquellen

Es sei $X$ eine Konstruktion mit $n$ Zeilen und $k$ Spalten und $\vec{Q}$ ein Zustand der Lichtquellen.
Dann laesst sich eine Eingabe fuer die zweite Zeile der Konstruktion gemaesz Definition 5 so definieren, dass sie genau aus $k$ Nullen und Einsen besteht, sodass die Einsen genau an den Positionen der Lichtquellen mit dem Zustand Eins stehen und der Rest durch Nullen befuellt wird.
Dann kann die Zeile 2 auf diese Eingabe mit einer Ausgabe $t$ reagieren, welche wiederum als Eingabe fuer die dritte Zeile benutzt wird.
Dieses iterative Verfahren wird fortgefuehrt, bis die Zeile $n-1$ eine Ausgabe geliefert hat.
Auf Basis dieser kann nun ausgewertet werden, welche der $l$ LEDs in Zeile $n$ aktiviert werden. Dies sind genau die LEDs mit einer Position, die in der letzen Ausgabe den Wert Eins hat.

### Bemerkung 9:

Das soeben beschriebene Verfahren sorgt dafuer, dass eine „Signal“ bzw. der Ausgabewert eines Bausteins stets nur in der direkt folgenden Zeile weiterverwendet werden kann, nicht jedoch in einer Zeiler weiter da runter.

### Bemerkung 10:

Analog zu dem beschriebenen Verfahren, kann man die $n-2$ Zeilen der Konstruktion mit Bausteinen auch als Funktionen $f_i : \{0,\ 1\}^k \longrightarrow \{0,\ 1\}^k$ interpretieren, wobei $i = 2,\ 3\ \dots n-1$. Dann koennte man die Bausteine auch als Funktionen $b : \{0,\ 1\}^2 \longrightarrow \{0,\ 1\}^2$ interpretieren, welche zusammen je die Funktion einer Zeile ergeben.
Dadurch koennte man die die Ausgabe der gesamten $n-2$ Zeilen fuer eine Eingabe $x$ sofort berechnen durch

$$(f_2 \circ\ f_3 \circ \cdots \circ f_{n-1})(x)$$

**Algorithmus 1 :** Calculate all Outputs

**Input :** Konstruktion $X$

**Output :** Ausgabe aller moeglichen Eingabenzustaende und den dazugehoerigen Ausgaben

**for** variation $\vec{Q}$ **in** all variations of size $q$ of $\{0,\ 1\}$ with repition **do**
  $x \leftarrow \ldots(x_1,\ \ldots\ ,\ x_k)$ where $x_{Q_i}$ is 1 iff $\vec{Q}$ is 1 at position $i$
  $t \leftarrow$ `calculate(`$x$`)`
  $\vec{L} \leftarrow (l_1,\ \ldots\ ,\ l_l)$ where $l_i$ is 1 iff t is one at position $L_i$
  ouput $\vec{Q}$ and $l$
**end for**

**Function** `calculate(`$x$`)`**:**
  **for** $i = 0$ ***to*** $n - 2$ **do**
    $n \leftarrow (0,\ 0,\ \ldots\ ,\ 0)$ `// Length` $k$
    $c \leftarrow 1$ `// Column`
    **while** $c \leq k - 1$ **do**
      $a \leftarrow$ entry of $X$ in row $i$ and column $c$
      $b \leftarrow$ entry of $X$ in row $i$ and column $(c + 1)$
      `// Check all cases of building blocks`
      **if** $a = X$ **then**
        continue
      **end if**
      **if** $a = B$ **then**
        $n[c] = x[c]$
        $n[c + 1] = x[c + 1]$
      **end if**
      **if** $a = W$ **then**
        $n[c] = \neg(x[c] \wedge x[c + 1])$
        $n[c + 1] = \neg(x[c] \wedge x[c + 1])$
      **end if**
      **if** $a = R$ **then**
        $n[c] = \neg x[c]$
        $n[c + 1] = \neg x[c]$
      **end if**
      **if** $a = r$ **then**
        $n[c] = \neg x[c + 1]$
        $n[c + 1] = \neg x[c + 1]$
      **end if**
      $c \leftarrow c + 2$
    **end while**
    $x \leftarrow n$
  **end for**
  **return** $x$

# 2. Loesungsvorschlag

Der Loesungsvorschlag folgt direkt aus Definition 8 und den vorherigen Bemerkungen und Definitionen.
Dazu werden fuer eine Konstruktion $X$ alle moeglichen Variationen der Zustaende der Lichtquellen durch iteriert. Fuer jeden dieser Zustaende wird eine Eingabe $x$ der Lange $k$ fuer die erste Zeile, wie beschrieben berechnet. Diese enthaelt Einsen genau an den Positionen der aktuell aktivierten Quellen.
Anschlieszend werden je iterativ die Ausgaben der Zeilen der Konstruktion fuer die aktuelle Eingabe berechnet. Dazu werden fuer jede Zeile die in Bemerkung 4 erlaeuterten logischen Formeln abhaengig vom Typ des Bausteins verwendet. Zur Erkennung der Bausteine wird dafuer in jeder Zeile der Konstruktion durch die Spalten iteriert, wobei zur naechsten Spalte gesprungen wird, falls ein $X$ gelesen wird, und zu uebernaechsten Spalte gesprungen wird, fall ein Baustein gefunden wurde. Dies wird wiederholt, bis die letzte Spalte uebersprungen wurde.

# 3. Analyse der Zeit- und Platzkomplexitaet der Loesung

Nachdem nun die Loesung vorgestellt wurde, folgt die Analyse der Zeit- und Komplexitaet.
Dabei wird davon ausgegangen, dass das Speichern von Wahrheitswerden und Ganzzahlen, sowie der verwendeten Zeichenketten eine konstante Platzkomplexitaet besitzt. Weiter wird angenommen, dass logische Operationen und Operationen auf Ganzzahlen eine konstante Zeitkomplexitaet besitzen.
Dadurch ergibt sich fuer einen Aufruf der Funktion „calculate“ die Zeitkomplexitaet

$$(n-2) \cdot \mathcal{O}(k) = \mathcal{O}(n \cdot k),$$

da die aeuszere Schleife genau $n-2$ mal und die innere Schleife maximal $k$ mal durchlaeuft und die weiteren Anweisung eine konstante Zeitkomplexitaet haben. Weiter ist die Platzkomplexitaet

$$\mathcal{O}(k),$$

da zu jedem Zeitpunkt maximal 2 Listen der Laenge $k$ existieren und die weiteren Variablen eine konstante Platzkomplexitaet aufweisen.

Dadurch laesst sich die gesamt Zeit- und Platzkomplexitaet ermitteln. Dabei wird die aeuszerste Schleife genau $2^q$ mal durchlaufen. Bei jedem Durchlauf wird die „calculate“ Funktion ausgefuehrt. Die anderen Operationen sind zu vernachlaessigen. Somit ergibt sich die Zeitkomplexitaet

$$2^q \cdot \mathcal{O}(n \cdot l) = \mathcal{O}(2^q \cdot n \cdot l).$$

Weiter existiert bei jedem Durchgang die Konstruktion der grosze $n \cdot k$ und die Listen der Laengen $q$, $k$ und $l$. Wegen $q,\ l \leq k$ folgt die Platzkomplexitaet

$$\mathcal{O}(k) + \mathcal{O}(n \cdot k) = \mathcal{O}(n \cdot k).$$

# 4. Implementierung

Die vorgestellte Loesung wurde in C++ implementiert. Dabei wurde ein zwei-dimensionaler Vektor an String zum Speichern der mittleren $n-2$ Zeilen verwendet. Weiter werden die Positionen der Quellen und LEDs in Vektoren an Integern gespeichert. Auszerdem werden die Ein- und Ausgaben der Zeilen der Konstruktion stets als Boolean-Vektor gespeichert.
Ein weiteres Implementierungsdetail besteht in der Berechnung aller $2^q$ moeglichen Zustaende der Quellen. Dazu wird via Bit-Manipulation zunaechst die Zahl $2^q$ berechnet, indem fuer die Zahl 1 $q$ Bits nach links geshiftet werden. Weiter wird ein Integer $i$ benutzt, um die aktuelle Eingabe / Variation darzustellen. Dieser laeuft von 0 bis $2^q - 1$. In jedem Durchgang wird anschlieszend geprueft, welche der Bits in der Binaerdarstellung von $i$ eins sind, um die aktuelle Variation zu erhalten.

# 5. Beispiele

Nun wird auf die Beispiele der BwInf Website eingegangen. Dabei werden je die $2^q$ Belegungen der Taschenlampen angegeben durch Elemente der Menge $\{0,\ 1\}^n$, wobei $q$ die Anzahl der Lichtquellen ist. Zu jeder Belegung wird dann die jeweilige Ausgabe der LEDs ebenfalls durch Nullen und Einsen angegeben.

## 5.1. Beispiel 1 - „nandu1.txt“

1. $(0,\ 0) \longrightarrow (1,\ 1)$
2. $(0,\ 1) \longrightarrow (1,\ 1)$
3. $(1,\ 0) \longrightarrow (1,\ 1)$
4. $(1,\ 1) \longrightarrow (0,\ 0)$

## 5.2. Beispiel 2 - „nandu2.txt“

1. $(0,\ 0) \longrightarrow (0,\ 1)$
2. $(0,\ 1) \longrightarrow (0,\ 1)$
3. $(1,\ 0) \longrightarrow (0,\ 1)$
4. $(1,\ 1) \longrightarrow (1,\ 0)$

## 5.3. Beispiel 3 - „nandu3.txt“

1. $(0,\ 0,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1)$
2. $(0,\ 0,\ 1) \longrightarrow (1,\ 0,\ 0,\ 0)$
3. $(0,\ 1,\ 0) \longrightarrow (1,\ 0,\ 1,\ 1)$
4. $(0,\ 1,\ 1) \longrightarrow (1,\ 0,\ 1,\ 0)$
5. $(1,\ 0,\ 0) \longrightarrow (0,\ 1,\ 0,\ 1)$
6. $(1,\ 0,\ 1) \longrightarrow (0,\ 1,\ 0,\ 0)$
7. $(1,\ 1,\ 0) \longrightarrow (0,\ 1,\ 1,\ 1)$
8. $(1,\ 1,\ 1) \longrightarrow (0,\ 1,\ 1,\ 0)$

## 5.4. Beispiel 4 - „nandu4.txt“

1. $(0,\ 0,\ 0,\ 0) \longrightarrow (0,\ 0)$
2. $(0,\ 0,\ 0,\ 1) \longrightarrow (0,\ 0)$
3. $(0,\ 0,\ 1,\ 0) \longrightarrow (0,\ 1)$
4. $(0,\ 0,\ 1,\ 1) \longrightarrow (0,\ 0)$
5. $(0,\ 1,\ 0,\ 0) \longrightarrow (1,\ 0)$
6. $(0,\ 1,\ 0,\ 1) \longrightarrow (1,\ 0)$
7. $(0,\ 1,\ 1,\ 0) \longrightarrow (1,\ 1)$
8. $(0,\ 1,\ 1,\ 1) \longrightarrow (1,\ 0)$
9. $(1,\ 0,\ 0,\ 0) \longrightarrow (0,\ 0)$
10. $(1,\ 0,\ 0,\ 1) \longrightarrow (0,\ 0)$
11. $(1,\ 0,\ 1,\ 0) \longrightarrow (0,\ 1)$

12. $(1,\ 0,\ 1,\ 1) \longrightarrow (0,\ 0)$
13. $(1,\ 1,\ 0,\ 0) \longrightarrow (0,\ 0)$
14. $(1,\ 1,\ 0,\ 1) \longrightarrow (0,\ 0)$
15. $(1,\ 1,\ 1,\ 0) \longrightarrow (0,\ 1)$
16. $(1,\ 1,\ 1,\ 1) \longrightarrow (0,\ 0)$

## 5.5. Beispiel 5 - „nandu5.txt“

1. $(0,\ 0,\ 0,\ 0,\ 0,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
2. $(0,\ 0,\ 0,\ 0,\ 0,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
3. $(0,\ 0,\ 0,\ 0,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
4. $(0,\ 0,\ 0,\ 0,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
5. $(0,\ 0,\ 0,\ 1,\ 0,\ 0) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
6. $(0,\ 0,\ 0,\ 1,\ 0,\ 1) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
7. $(0,\ 0,\ 0,\ 1,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
8. $(0,\ 0,\ 0,\ 1,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
9. $(0,\ 0,\ 1,\ 0,\ 0,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
10. $(0,\ 0,\ 1,\ 0,\ 0,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
11. $(0,\ 0,\ 1,\ 0,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
12. $(0,\ 0,\ 1,\ 0,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
13. $(0,\ 0,\ 1,\ 1,\ 0,\ 0) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
14. $(0,\ 0,\ 1,\ 1,\ 0,\ 1) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
15. $(0,\ 0,\ 1,\ 1,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
16. $(0,\ 0,\ 1,\ 1,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
17. $(0,\ 1,\ 0,\ 0,\ 0,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
18. $(0,\ 1,\ 0,\ 0,\ 0,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
19. $(0,\ 1,\ 0,\ 0,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
20. $(0,\ 1,\ 0,\ 0,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
21. $(0,\ 1,\ 0,\ 1,\ 0,\ 0) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
22. $(0,\ 1,\ 0,\ 1,\ 0,\ 1) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
23. $(0,\ 1,\ 0,\ 1,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
24. $(0,\ 1,\ 0,\ 1,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
25. $(0,\ 1,\ 1,\ 0,\ 0,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
26. $(0,\ 1,\ 1,\ 0,\ 0,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 0)$
27. $(0,\ 1,\ 1,\ 0,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
28. $(0,\ 1,\ 1,\ 0,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
29. $(0,\ 1,\ 1,\ 1,\ 0,\ 0) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
30. $(0,\ 1,\ 1,\ 1,\ 0,\ 1) \longrightarrow (0,\ 0,\ 1,\ 0,\ 0)$
31. $(0,\ 1,\ 1,\ 1,\ 1,\ 0) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$

32. $(0,\ 1,\ 1,\ 1,\ 1,\ 1) \longrightarrow (0,\ 0,\ 0,\ 1,\ 1)$
33. $(1,\ 0,\ 0,\ 0,\ 0,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
34. $(1,\ 0,\ 0,\ 0,\ 0,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
35. $(1,\ 0,\ 0,\ 0,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
36. $(1,\ 0,\ 0,\ 0,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
37. $(1,\ 0,\ 0,\ 1,\ 0,\ 0) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
38. $(1,\ 0,\ 0,\ 1,\ 0,\ 1) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
39. $(1,\ 0,\ 0,\ 1,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
40. $(1,\ 0,\ 0,\ 1,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
41. $(1,\ 0,\ 1,\ 0,\ 0,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
42. $(1,\ 0,\ 1,\ 0,\ 0,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
43. $(1,\ 0,\ 1,\ 0,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
44. $(1,\ 0,\ 1,\ 0,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
45. $(1,\ 0,\ 1,\ 1,\ 0,\ 0) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
46. $(1,\ 0,\ 1,\ 1,\ 0,\ 1) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
47. $(1,\ 0,\ 1,\ 1,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
48. $(1,\ 0,\ 1,\ 1,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
49. $(1,\ 1,\ 0,\ 0,\ 0,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
50. $(1,\ 1,\ 0,\ 0,\ 0,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
51. $(1,\ 1,\ 0,\ 0,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
52. $(1,\ 1,\ 0,\ 0,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
53. $(1,\ 1,\ 0,\ 1,\ 0,\ 0) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
54. $(1,\ 1,\ 0,\ 1,\ 0,\ 1) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
55. $(1,\ 1,\ 0,\ 1,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
56. $(1,\ 1,\ 0,\ 1,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
57. $(1,\ 1,\ 1,\ 0,\ 0,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
58. $(1,\ 1,\ 1,\ 0,\ 0,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 0)$
59. $(1,\ 1,\ 1,\ 0,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
60. $(1,\ 1,\ 1,\ 0,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
61. $(1,\ 1,\ 1,\ 1,\ 0,\ 0) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
62. $(1,\ 1,\ 1,\ 1,\ 0,\ 1) \longrightarrow (1,\ 0,\ 1,\ 0,\ 0)$
63. $(1,\ 1,\ 1,\ 1,\ 1,\ 0) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$
64. $(1,\ 1,\ 1,\ 1,\ 1,\ 1) \longrightarrow (1,\ 0,\ 0,\ 1,\ 1)$

# 6. Quellcode

Nun folgt der Quellcode des wichtigsten Teils Loesung in C++.

## 6.1. Durchgehen durch alle moeglichen Eingaben fuer die Lichtquellen

```
for(int i = 0; i < amount_total; i++) {
      // The current variation is given by the binary representation of i.
      // start will store an input for the building blocks stores in blocks;
      // e.i., every column will start with 0 or 1. Ones can only occour if
      // there is a lamp in that column and that one is enabled in the current variation.
      vector<bool> start(columns, 0);

      // Calculate the start vector (i.e. the input for the builduing blocks)
      // using the binary representation of i.
      for (int j = 0; j < amount_inputs; j++) {
          // Check if the j-th bit of i is 1 from the right
          if (i & (1 << (amount_inputs - j - 1))) {
              // If so, set the j-th lamp to 1.
              // That is, set the position of the j-th lamp to 1 in the start vector
              start[inputs[j]] = 1;
          }
      }

      // Calculate the resulting output using the current input (start).
      vector<bool> result = create_result(blocks, columns, rows, start);

      // Ausgabe
      [...]
}
```

## 6.2. Berechnung der Ausgabe der LEDs bei einer bestimmten Eingabe der Lichtquellen

```
// Function computing the resulting boolean vector created by inputing
// a given boolean vector input into the building-blocks construct given.
vector<bool> create_result(vector<vector<string>> &blocks,
                           int columns, int rows, vector<bool> &start) {
  // Track the current input row using the current_row boolean vector
  // of length (columns) and iterativle construct the next one.
  vector<bool> current_row = start;

  // Iterate through all rows of the building blocks
  // and compute the next input row using the current one (current_row)
  for (int row = 0; row < rows - 2; row++) {
    // Store the input row in next_row
    vector<bool> next_row(columns, 0);

    // Iterate through the columns of the current row
    // and construct the next input row
    int column = 0;

    // Loop as long as the current column is smaller then columns - 1,
    // because the program will try to access the symbol the the right of column,
    // since the building blocks only come in pairs of two symbols.
    while (column < columns - 1) {
      // If the current column does not contains a building block,
      // skip to the next column.
      if (blocks[row][column] == X) {
        column++;
        continue;
      }

      // Receive the current building block consisting of two symbols
      string s1 = blocks[row][column];
      string s2 = blocks[row][column + 1];
```

```
34        // Check for the different types of building blocks:
          // blue, white, and red
36        // For each one: Construct the next input row.

38        // 1. Case: Blue building block
          if (s1 == B) {
40          // A blue block correspond to passing the current state down to the next one
            next_row[column] = current_row[column];
42          next_row[column + 1] = current_row[column + 1];
          // 2. Case: White building block
44        } else if (s1 == W) {
            // A white block enables both columns iff (if and only if)
46          // less then two columns are "enabled" in the current rows.
            // That is iff not both columns are "enabled".
48          next_row[column] = !(current_row[column] && current_row[column + 1]);
            next_row[column + 1] = !(current_row[column] && current_row[column + 1]);
50        // 3. Case: Red building block
          } else if (s1 == r || s1 == R) {
52          // 3.1: The first symbol is R (upper case) and the second is r (lower case)
            // In that case both columns of the next row are enabled iff the first
54          // of the two colums is "disabled".
            if (s1 == R) {
56            next_row[column] = !current_row[column];
              next_row[column + 1] = !current_row[column];
58          // 3.2: the first symbol is r (lower case) and the second is R (upper case)
            // In that case the both columns of the next row are enabled iff the second
60          // of the two colums is "disabled".
            } else {
62            next_row[column] = !current_row[column + 1];
              next_row[column + 1] = !current_row[column + 1];
64          }
          }
66
          // Increase the column by 2 to check for the next building block
68        column += 2;
        }
70
        // Set the next input row to be the one just constructed
72      current_row = next_row;
      }
74
      // Return the next input row computes last,
76    // such that the enabled/disabled LEDs can be computed.
      return current_row;
78  }
```