Matthias Rüster
Lena Kristin Zander

Bundeswettbewerb Informatik: Aufgabe 2

Lösungsidee

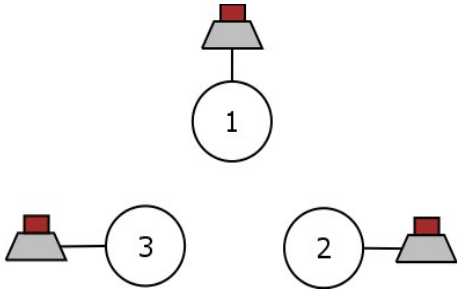
Zu jeder Schaltung mit n Lampen gibt es 2^n-1 Ausgangssituationen, die Lösung (alle Lampen sind angeschaltet) ist dabei ausgeschlossen.

Die maximale Anzahl der Verbindungen beträgt n^2 (mit den regulären Verbindungen: jeder Taster verändert seine eigene Lampe).

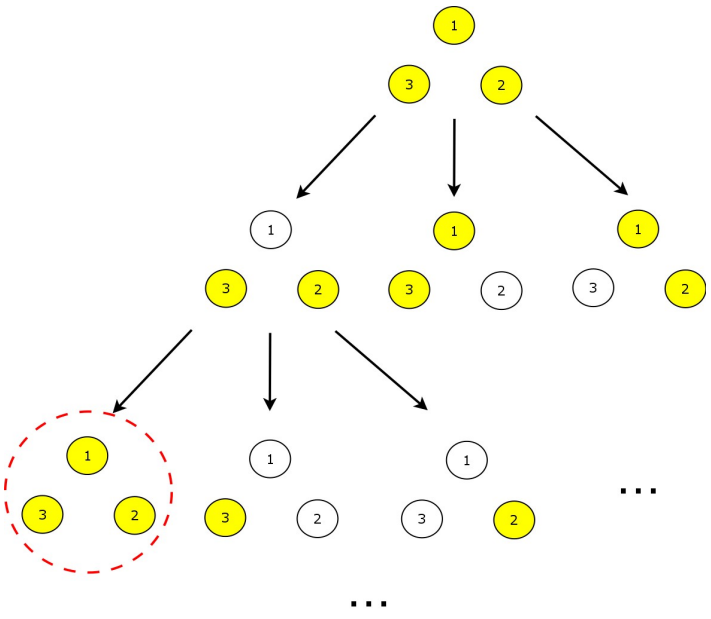
Variantenbaum

All diese Ausgangssituationen lassen sich in einem Variantenbaum darstellen. Die Lösung bildet dabei die Wurzel. Der Variantenbaum wird durch eine rekursive Funktion erzeugt.

Beispielschaltung:



Für diese Schaltung würde folgender Variantenbaum entstehen:



Von jeder Kombination aus hat man n Möglichkeiten den Zustand der Lampen zu verändern. Erreicht man eine Kombination, die man bereits gefunden hat, endet der Zweig an dieser Stelle (siehe markierter Bereich).

Lösen einer beliebigen Ausgangssituation

Statt von einer Schaltung ausgehend eine Tastenfolge zu finden, die alle Lampen anschaltet, kann man diesen Vorgang auch umdrehen. Beginnend bei der Lösung wird der Variantenbaum solange durchlaufen, bis die Ausgangssituation erreicht wurde. Wenn man den gesamten Variantenbaum durchlaufen hat, ohne die gesuchte Ausgangssituation gefunden zu haben, ist diese nicht lösbar.

Schaltung auf Brauchbarkeit prüfen

Um eine Schaltung darauf zu prüfen, ob sie brauchbar ist, durchläuft man den Variantenbaum und zählt dabei die gefundenen Ausgangssituationen. Hat man 2^n -1 verschiedene gefunden, ist die Schaltung brauchbar. Sind weniger gefunden worden, so ist die Schaltung nicht brauchbar.

Zufällige Schaltungen

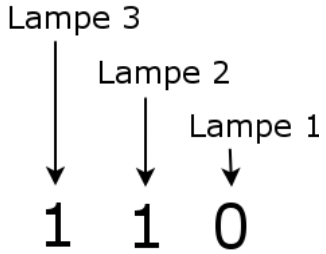
Per Zufall gewählt werden kann die Anzahl der Lampen und Verbindungen. Des Weiteren werden Taster zufällig mit Lampen verbunden.

Der Benutzer ist jedoch in der Lage die jeweiligen Werte selbst zu bestimmen.

Eine zufällig generierte Schaltung wird darauf geprüft, ob sie brauchbar ist. Ist sie unbrauchbar, wird eine weitere generiert.

Bitmasken

Der Zustand der Lampen wird durch eine Variable dargestellt, bei der die einzelnen Bits für die jeweiligen Lampen stehen:



In diesem Fall sind die Lampen 2 und 3 angeschaltet.

Die Taster werden als Bitmasken abgespeichert:

$$\begin{aligned} \text{Taster } 1 &\rightarrow 001 \\ \text{Taster } 2 &\rightarrow 010 \\ \text{Taster } 3 &\rightarrow 100 \end{aligned}$$

Jedes gesetzte Bit in einer Bitmaske stellt die Verbindung zwischen Taster und Lampe dar.

So ist es möglich die Zustände der Lampen mit Hilfe der XOR-Verknüpfung zu ändern:

$$\begin{array}{ccccc} \text{Zustand}_{alt} & \oplus & \text{Taster } x & = & \text{Zustand}_{neu} \quad x \in [1,n] \\ 110_2 & \oplus & 010_2 & = & 100_2 \end{array}$$

Da die XOR-Verknüpfung kommutativ und assoziativ ist[1], besteht eine Tastenfolge aus maximal *n* Tastenbetätigungen.

Folgende Regeln gelten:

$$\begin{array}{lrcl} I: & A \oplus A & = & 0 \\ II: & A \oplus 0 & = & A \\ III: & A \oplus B & = & B \oplus A \\ IV: & (A \oplus B) \oplus C & = & A \oplus (B \oplus C) \end{array}$$

A, B und C seien Bitmasken.

Man kann also jede Tastenfolge, die aus mehr als *n* Tastenbetätigungen besteht, mit den obigen Formeln vereinfachen:

$$\begin{array}{lllllllll|l} A & \oplus & B & \oplus & A & \oplus & C & \oplus & A & III \\ A & \oplus & B & \oplus & C & \oplus & A & \oplus & A & I \\ A & \oplus & B & \oplus & C & \oplus & 0 & & & II \\ A & \oplus & B & \oplus & C & & & & & \end{array}$$

1 Axiome der Schaltalgebra (http://en.wikipedia.org/wiki/Boolean_algebra_%28logic%29#Axioms). Ein Beweis kann auch über Wahrheitstabellen erfolgen.

Programm-Dokumentation

Eine Schaltung wird als eine **game**-Struktur abgespeichert. Das Element **lamps** der **game**-Struktur enthält die Bitmaske für den aktuellen Zustand der Lampen. Des Weiteren enthält die **game**-Struktur das Element **button**, dass ein Array von Bitmasken ist, welche die Tasten darstellen. Die Anzahl der Tasten wird in dem Element **count** der **game**-Struktur gespeichert..

Eine Schaltung wird mit Hilfe der Funktion **read_initial_situation** eingelesen. Diese Funktion bekommt ein Dateinamen als Parameter übergeben, liest die Schaltung ein und speichert sie in einer **game**-Struktur. Die Datei mit der Schaltung, die eingelesen werden soll, muss wie folgt ausssehen:

```
7
1: 7 1 2 aus
2: 1 2 3 aus
3: 2 3 4 an
4: 3 4 5 aus
5: 4 5 6 an
6: 5 6 7 aus
7: 6 7 1 an
```

Die erste Zeile der Datei enthält die Anzahl der Lampen/Taster. Danach werden die Verbindungen des Tasters und der Zustand der dazugehörigen Lampe definiert: Beispielsweise hat der erste Taster Verbindungen zu den Lampen 1, 2 und 7 und der Zustand seiner Lampe ist „aus“.

Gibt man den Zustand der zum Taster zugehörigen Lampe nicht an, wird dieser standardmäßig auf „aus“ gesetzt.

Das Lösen einer Ausgangssituation übernimmt die Funktion **game_solve**. Die Funktion bekommt die **game**-Struktur als Parameter übergeben und ruft wiederum die rekursive Funktion **solve** auf. Die Funktion **solve** durchläuft den Variantenbaum und gibt den Wert 1 zurück, sobald die Ausgangssituation erreicht wurde. Ist die Ausgangssituation jedoch nicht gefunden worden, gibt es auch keine Tastenfolge, die zu einer Lösung führt.

Mit der Funktion **game_check** kann die Schaltung auf Brauchbarkeit geprüft werden. Auch diese Funktion bekommt eine **game**-Struktur als Parameter übergeben und ruft die rekursive Funktion **check** auf. Diese Funktion durchläuft den Variantenbaum und merkt sich alle besuchten Ausgangssituationen in einem Array. Ist der gesamte Variantenbaum durchlaufen, prüft die Funktion **game_check**, ob 2^n-1 Ausgangssituation ermittelt werden konnten. Andernfalls ist die Schaltung nicht brauchbar.

Alle besuchten Ausgangssituationen müssen gespeichert werden. Zum Speichern aller Ausgangssituationen benötigt man demnach $2^n - 1$ Bits = $(2^n - 1) / 8$ Bytes.

Die maximale Rekursionstiefe zum Durchlaufen des Variantenbaums beträgt $n + 1$.

Die maximale Anzahl der Lampen ist somit abhängig von dem verfügbaren Arbeitsspeicher (RAM). Berechnet werden kann diese wie folgt:

$$\text{Speicher}_{benötigt} = \frac{2^{Lampen}}{8} + \text{X}$$

$$\text{Lampen}_{Max} \approx \log_2 \left[8 \cdot \left(\text{Speicher}_{verfügbar} - \text{X} \right) \right]$$

Der Speicher ist in Bytes angegeben.

X steht für den Speicherbedarf, der für die Ausführung des Programms (z. B. für lokale Variablen und die Rekursion) benötigt wird.

Bei 1 GB = 2^{30} Bytes verfügbarem Arbeitsspeicher können Schaltungen mit maximal 33 Lampen berechnet werden.

Mit der Funktion **`create_random`** kann eine zufällige Schaltung generiert werden. Sie bekommt zwei Parameter übergeben. Der erste Parameter bestimmt, wie viele Lampen die Schaltung umfassen soll. Der zweite Parameter legt die Anzahl der Verbindungen zusätzlich zu den Regulären fest. Beide Werte werden zufällig bestimmt, sofern der Benutzer keine weiteren Angaben macht.

Zufällig generierte Schaltungen werden dann wieder mit Hilfe der Funktion **`game_check`** auf Brauchbarkeit geprüft. Sollte die generierte Schaltung nicht brauchbar sein, wird solange eine neue generiert, bis eine brauchbare gefunden wurde.

Programm-Ablaufprotokoll

Informationen darüber, welche Parameter dem Programm übergeben werden können, erhält der Benutzer, indem er das Programm mit dem Parameter **`-h`** aufruft:

```
$ ./aufgabe2 -h
Verwendung: aufgabe2 [Optionen] [Dateiname]

Einlesen einer bestimmten Schaltung:
    -f Dateiname      Schaltung auf Brauchbarkeit pruefen
    -s Dateiname      Tastenfolge zum Loesen der Schaltung ermitteln

Beim Generieren der zufaelligen Schaltungen
koennen folgende Parameter festgelegt werden:
    -l Lampen         Anzahl der Lampen festlegen
    -c Verbindungen   Anzahl der Verbindungen festlegen
```

Möchte der Benutzer eine zufällige Schaltung generieren, dann kann er das Programm auch ohne Parameter aufrufen. Die Anzahl der Lampen und Verbindungen werden dann zufällig gewählt. Ist eine brauchbare Schaltung gefunden worden, wird diese auf dem Bildschirm ausgegeben:

```
$ ./aufgabe2
Brauchbare Schaltung gefunden:
Lampenanzahl: 10
Schaltung:
 1: 1 3
 2: 2 5 6 7
 3: 1 3 6 8
 4: 4 6
 5: 5 6 9
 6: 3 6 7
 7: 7 8
 8: 3 8
 9: 3 6 9
10: 10
```

Wird das Programm mit dem Parameter **`-l 3`** und **`-c 2`** aufgerufen, werden auch nur brauchbare Schaltungen erzeugt, die 3 Lampen mit 2 Verbindungen besitzen:

```
$ ./aufgabe2 -l 3 -c 2
Brauchbare Schaltung gefunden:
Lampenanzahl: 3
Schaltung:
 1: 1 3
 2: 2
 3: 2 3

$ ./aufgabe2 -l 3 -c 2
Brauchbare Schaltung gefunden:
Lampenanzahl: 3
Schaltung:
 1: 1 2
 2: 2 3
 3: 3
```

Möchte man eine bestimmte Schaltung auf Brauchbarkeit prüfen, kann man das Programm mit **-f *Dateiname*** aufrufen. Die Datei **beispiel2.txt** enthält beispielsweise die zweite Schaltung der Aufgabenstellung:

```
9
1: 1 5 4 2
2: 1 2 3
3: 2 3 6 5
4: 1 4 7
5: 6 8 5 2 4
6: 9 6 3
7: 8 7 4 5
8: 9 8 7
9: 8 6 5 9
```

Um diese auf Brauchbarkeit zu prüfen, ruft man das Programm folgendermaßen auf:

```
$ ./aufgabe2 -f beispiel2.txt
Lampenanzahl: 9
Schaltung:
 1: 1 2 4 5
 2: 1 2 3
 3: 2 3 5 6
 4: 1 4 7
 5: 2 4 5 6 8
 6: 3 6 9
 7: 4 5 7 8
 8: 7 8 9
 9: 5 6 8 9

Die Schaltung ist brauchbar.
```

Es waren also alle möglichen Ausgangssituationen im Variantenbaum enthalten.

Die erste Beispielschaltung ist in der Datei **beispiel1.txt** abgespeichert.

Um eine Tastenfolge zu finden, die diese löst, wird das Programm mit dem Parameter **-s beispiel1.txt** aufgerufen:

```
$ ./aufgabe2 -s beispiel1.txt
Lampenanzahl: 7
Schaltung:
 1 (aus): 1 2 7
 2 (aus): 1 2 3
 3 ( an): 2 3 4
 4 (aus): 3 4 5
 5 ( an): 4 5 6
 6 (aus): 5 6 7
 7 ( an): 1 6 7

Die Schaltung kann mit folgender Tastenfolge geloest werden:
1 7 3 2
```

Um diese auch noch auf Brauchbarkeit zu prüfen, ruft man das Programm wieder mit dem Parameter **`-f`** auf:

```
$ ./aufgabe2 -f beispiel1.txt
Lampenanzahl: 7
Schaltung:
 1: 1 2 7
 2: 1 2 3
 3: 2 3 4
 4: 3 4 5
 5: 4 5 6
 6: 5 6 7
 7: 1 6 7

Die Schaltung ist brauchbar.
```