Aufgabe 4: Zara Zackigs Zurückkehr

Teilnahme-ID: 62454

Bearbeiter dieser Aufgabe:
Philip Gilde

21. April 2022

Inhaltsverzeichnis

1 Lösungsidee

Von den gegebenen n Karten mit jeweils m Bits werden p Karten gesucht, so dass das exklusive Oder (im Folgenden als XOR abgekürzt) von $p-1$ Karten gleich der p. Karte ist.

$$\begin{aligned} & k_1 \oplus k_2 \oplus \ldots \oplus k_{p-1} = k_p && \mid \oplus k_p \\ \leftrightarrow \quad & k_1 \oplus k_2 \oplus \ldots \oplus k_{p-1} \oplus k_p = 0 \end{aligned}$$

Diese Gleichung lässt sich zu jeder der p Karten umstellen. Es werden also p Karten gesucht, deren XOR gleich einer Karte mit m Nullen ist. Dieses Problem lässt sich umformulieren zu einem linearen Gleichungssystem im Galois-Feld $GF(2)$. Dieses besteht nur aus den beiden Elementen 0 und 1. Die Addition im Feld entspricht dem XOR, die Multiplikation einem UND. Weil in dem Feld die Multiplikation und Addition definiert ist, können diese Operationen auch in Form von linearer Algebra verwendet werden. Die gemischten Karten entsprechen der Matrix $K \in GF(2)^{n \times m}$. K_n ist dabei die n-te Karte und $K_{n,m}$ das m-te Bit der n-ten Karte. Gesucht wird der Vektor $v \in GF(2)^n$, so dass dieses lineare Gleichungssystem gilt:

$$\begin{aligned} K_{1,1}v_1 + K_{2,1}v_2 + \ldots + K_{n,1}v_n &= 0 \\ K_{1,2}v_1 + K_{2,2}v_2 + \ldots + K_{n,2}v_n &= 0 \\ &\ldots \\ K_{1,m}v_1 + K_{2,m}v_2 + \ldots + K_{n,m}v_n &= 0 \\ K^T v &= 0 \end{aligned}$$

Dabei bestimmt v_n, ob die n-te Karte zu den gesuchten Karten gehört.
Die Menge von Vektoren, die sich oben für v einsetzen lassen, wird als Nullraum oder Kern der Matrix K^T bezeichnet. In $GF(2)$ kann ein Vektor nicht skaliert werden, weil er nur mit entweder 0 oder 1 multipliziert werden kann. Somit besteht der Nullraum aus allen möglichen Kombinationen von Summen der Basisvektoren. Wenn r der Rang von K^T ist, dann ist $q = n - r$ die Anzahl der Basisvektoren des Nullraums [1, S. 63]. Der Rang von K^T ist die Anzahl linear unabhängiger Zeilen beziehungsweise Spalten

(diese beiden Werte sind gleich). Wenn, wie in der ursprünglichen Aufgabe, $n < m$, dann ist der Rang in der Regel $n-1$, denn nur eine Karte, die Wiederherstellungskarte, ist linear abhängig von den anderen. Die Wahrscheinlichkeit, dass h Karten mit jeweils b Bits voneinander linear unabhängig sind, ist, wenn diese zufällig und erzeugt sind und Nullen und Einsen gleich wahrscheinlich sind, wovon der Einfachheit halber ausgegangen wird, nach [2] gegeben durch

$$P(\text{"alle unabhängig"}) = \prod_{i=1}^{h}(1 - 2^{i-1-b})$$

Diese ist für $h = n-1 = 110$ und $b = m = 128$ hoch genug, um den anderen Fall zunächst zu vernachlässigen. Somit sind alle bis auf eine der 111 Karten linear unabhängig voneinander. Der Nullraum besteht damit aus nur $q = n - (n-1) = n - n + 1 = 1$ Vektor. Dieser muss an den Stellen der 11 echten Karten 1 und sonst überall 0 sein. Somit haben wir die echten Karten gefunden.

Um die Basisvektoren zu finden, wird das in [3] beschriebene Verfahren verwendet. Dabei wird zuerst K^T mit der Identitätsmatrix zu $\left[\frac{K^T}{I}\right]$ erweitert. $\left[\frac{K^T}{I}\right]^T = [K|I]$ wird mithilfe des Gauß-Jordan-Algorithmus in Stufenform gebracht, was dann in der sich in Stufenform befindenden Matrix $\left[\frac{B}{C}\right]^T$ resultiert. Die Matrix $\left[\frac{B}{C}\right]$ befindet sich in Spaltenstufenform. Die Basis des Nullraums bilden die Spalten von C, deren entsprechende Spalten in B Null sind. Das lässt sich folgendermaßen begründen: Die elementaren Reihentransformationen der Transponierten, die beim Gauß-Jordan-Algorithmus durchgeführt werden, entsprechen elementaren Spaltentransformationen, welche einer Multiplikation mit einer Matrix P entsprechen. Diese setzt sich als Produkt der einzelnen Schritte zusammen. Bei den Schritten handelt es sich immer um die Addition einer Spalte zu einer anderen, was der Multiplikation mit einer Elementarmatrix entspricht. Beispielsweise würde eine Multiplikation mit der Matrix

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

die erste Spalte zur zweiten addieren. Da es sich um Elementarmatrizen handelt, sind diese immer invertierbar. Somit ist P als Produkt von invertierbaren Matrizen auch invertierbar.
Es gilt also $\left[\frac{K^T}{I}\right] P = \left[\frac{B}{C}\right]$. Daraus folgt $IP = P = C$ und $K^T P = K^T C = B$.

$$\begin{aligned}
& K^T C = B && | \cdot C^{-1} \\
\leftrightarrow \quad & K^T = BC^{-1} && | \cdot v \\
\leftrightarrow \quad & K^T v = BC^{-1} v && |\text{Es wird } C^{-1}v = w \text{ gesetzt} \\
& = Bw && |\text{Damit } v \text{ zum Nullraum gehört, muss gelten:} \\
& = 0
\end{aligned}$$

Weil alle Spalten in B, die nicht Null sind, linear unabhängig voneinander sind (das ist eine Folge der Spaltenstufenform), gilt $Bw = 0$ nur wenn die Einträge von w, die nicht Null sind, den Nullspalten von B entsprechen. Die Basis der Vektoren w, für die $Bw = 0$ gilt, sind also die verschiedenen Vektoren, die eine Eins bei einer Nullspalte von B und sonst überall Nullen haben. Da $C^{-1}v = w \leftrightarrow v = Cw$ definiert wurde, sind die Spalten von C, die den Nullspalten von B entsprechen, die Basis des Nullraums.
Der Gauß-Jordan-Algorithmus bringt eine (erweiterte) Matrix in reduzierte Stufenform. Das bedeutet, dass das erste nicht-Null-Element in jeder Reihe auch das einzige nicht-Null-Element in seiner jeweiligen Spalte ist. Dadurch kann diese Reihe nicht als Linearkombination der anderen Reihen dargestellt werden und ist somit linear unabhängig. Das erste nicht-Null-Element einer Reihe befindet sich außerdem immer weiter hinten als das der vorhergehenden Reihen. Das alles wird bewerkstelligt, in dem nacheinander verschiedene Reihen zueinander addiert werden. In dem man eine Reihe, deren erstes Element noch nicht als einziges Eins ist, zu den anderen Reihen addiert, in denen dieses Element auch Eins ist, macht man

aus diesen Null. Das ganze Verfahren soll an einem Beispiel illustriert werden:

$$K = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

$$[K|I] = \left[\begin{array}{cccccc|cccccc} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Um die Stufenform zu erreichen, muss zuerst in der transformierten Matrix M $M_{1,1} = 1$ sein. Dafür wird eine Reihe, deren erstes Element 1 ist, zur ersten Reihe addiert. Hier wird das mit der zweiten Reihe getan:

$$\left[\begin{array}{cccccc|cccccc} 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Damit nun nur das erste Element der ersten Spalte 1 ist, wird die erste Reihe zu jeder anderen Reihe addiert, deren erstes Element 1 ist:

$$\left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Die zweite Spalte hat schon die richtige Form, die dritte hingegen nicht. Also wird die dritte Reihe zu jeder Reihe addiert, deren drittes Element 1 ist:

$$\left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

So wird auch in der vierten und fünften Spalte weitergemacht:

$$\left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \rightarrow \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{cccccc|cccccc} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right]$$

Die Matrix befindet sich jetzt in Stufenform, es handelt sich um $\left[\frac{B}{C}\right]^T$. Durch Transponieren erhält man:

$$\left[\frac{B}{C}\right] = \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

Weil nur die letzte Spalte von B Null ist, ist die letzte Spalte von C die Basis des Nullraums. Die Öffnungskarten sind also die letzten drei:

$$\left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 \end{array}\right]$$

Wie man sieht, ist jede der Karten das XOR der anderen beiden, und die Lösung somit korrekt.

Wenn $q > 1$ ist, kann es sein, dass keiner der Basisvektoren des Nullraums 11 Einsen beinhaltet, sondern eine Linearkombination dieser. In diesem Fall wird jede der 2^q Kombinationen der Basisvektoren durchprobiert, bis eine davon 11 Eisen enthält.

Die richtige Karte für das s-te Haus kann gefunden werden, in dem man die Karten aufsteigend sortiert und zuerst die s-te und dann die $s+1$-te Karte ausprobiert. Wenn die Sicherungskarte kleiner als die Öffnungskarte des s-ten Hauses ist, liegt sie davor im Stapel und die Öffnungskarte an der Stelle $s+1$. Wenn sie größer ist, dann liegt sie dahinter im Stapel und die Öffnungskarte an Stelle s.

Das Verfahren stößt an seine Grenzen, wenn $n \gg m$ ist. Der Rang von K^T ist dann nämlich höchstens m, wodurch der Nullraum $q = n - m$ Basisvektoren hat. Diese Basisvektoren müssen nun nicht zwangsweise einen mit 11 Einsen beinhalten, dieser könnte auch eine Linearkombination der anderen Basisvektoren sein. Wenn das der Fall ist, müsste man alle 2^q Kombinationen von Basisvektoren durchprobieren, bis eine davon 11 Einsen beinhaltet, was für die Beispieleingabe auf der BwInf-Webseite mit 161 Karten zu je 128 Bit $2^q = 2^{n-m} = 2^{161-128} = 2^{33} = 8.589.934.592$ Kombinationen sind. Diese lassen sich nicht wirklich in überschaubarer Zeit durchprobieren.

2 Umsetzung

Die Lösung wurde in Python 3.10 umgesetzt. Dabei wurde die Bibliothek `NumPy` verwendet, um die Matrizen darzustellen. Für eine Fortschrittsanzeige beim Zeitaufwendigen durchprobieren der 2^q Linearkombinationen wurde die Bibliothek `progressbar2` verwendet. Wenn das Programm aufgerufen wird, muss zuerst der Dateipfad zur Eingabedatei im Format wie auf der BwInf-Webseite eingegeben werden. Dann werden die echten Karten gesucht und ausgegeben.

3 Laufzeit

Der Gauß-Jordan-Algorithmus hat für eine erweiterte Matrix mit u Spalten in der linken und v Spalten in der rechten Matrix und w Reihen eine Laufzeitordnung von $\mathcal{O}(wu(u+v))$, denn für jede der u Spalten der linken Matrix muss eine Reihe zu allen w anderen Reihen XOR-t werden, also mit allen $u+v$ Elementen jeder Reihe. Weil die rechte Matrix eine Identitätsmatrix ist, hat diese so viele Reihen wie Spalten, also w Reihen und w Spalten. Somit ist $v = w$ und die Laufzeitordnung $\mathcal{O}(wu(u+w)) = \mathcal{O}(wu^2 + uw^2)$. Die Matrix K^T, für die der Algorithmus durchgeführt wird, hat die Dimensionen $n \times m$. Die Laufzeit ist somit $\mathcal{O}(mn^2 + nm^2)$. Wenn keiner der gefundenen Basisvektoren die richtige Anzahl an Einsen enthält, müssen alle 2^q Kombinationen durchprobiert werden. Dieser Schritt hat dann also die Laufzeit 2^q.

4 Beispiele

Die Eingabedateien wurden der Übersichtlichkeit halber hier weggelassen, sie sind im Ordner `beispiele` zu finden. Wie erklärt, konnten die schweren Eingaben nicht gelöst werden.
Beispiel 1:
Eingabe:

```
beispiele/stapel0.txt
```

Ausgabe:

```
Echte Karten:
[[0 0 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 0 0 1]
 [1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 1 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 1]
 [1 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 0 0 0]
 [1 0 1 0 1 1 0 0 1 1 1 1 1 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 0 0 0 0 0 0]
 [1 0 1 1 1 0 0 0 0 1 1 0 0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0]]
```

Beispiel 2:
Eingabe:

```
beispiele/stapel1.txt
```

Ausgabe:

```
Echte Karten:
[[0 0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 0]
 [1 1 0 1 0 0 1 1 0 1 0 1 1 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1 0 1 1 1]
 [0 0 1 1 0 1 0 0 0 0 1 0 1 0 1 0 0 1 0 0 0 0 0 1 1 1 1 0 1 0 0 1 0]
 [1 1 1 1 0 0 1 1 1 0 1 0 1 1 0 0 1 0 0 1 0 0 0 0 0 1 0 1 1 1 1 1 0]
 [0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 0 1 0]
 [1 1 1 1 0 1 1 1 1 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 1 1 0]
 [0 0 1 0 0 0 1 1 1 0 0 1 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1 0 0 0 1 1]
 [1 1 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 0 0 0 0 0 1 0 1 1 1 0 1 0 0]
 [0 0 0 1 0 0 0 1 1 1 0 1 0 0 1 1 0 0 0 1 1 1 1 1 0 1 1 0 0 1 0 0]]
```

Beispiel 3:
Eingabe:

```
beispiele/stapel2.txt
```

Ausgabe:

```
Echte Karten:
[[0 1 1 0 1 0 1 1 1 0 1 0 0 0 1 1 0 1 1 1 0 1 0 0 0 1 1 0 0 0 0 1 1 1 0 0
  0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 1 1 0 0 1 1 0
  1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 0 1 0 1 1 0 1 1 1 1 0 0 0 0 1 1 1 1 0 1
  1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1]
 [0 0 1 0 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 1 0 1 0 1 1 0 1 1 1 1 0 0 1 0 0 1
  1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 0
  0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 1 0 1 1 0 1 1 0 0 1 0 1 0 1 1 1 0 1 0 0 1
  0 0 0 0 1 0 1 1 1 0 0 0 1 1 0 1 0 0 0 1]
 [1 0 1 0 1 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 1 0 0
  1 1 0 0 0 1 1 0 0 1 1 1 0 0 1 0 1 0 1 1 0 1 1 1 1 1 0 1 1 0 0 0 1 1 1 1
  1 1 1 0 1 1 1 1 1 1 0 1 0 0 0 1 1 1 1 1 1 1 0 1 0 0 0 0 0 0 1 0 1 1 0 1 1 0
  1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0 1 1 1 1 1 0]
 [1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 1 1 0 0 0 0 0
  0 0 0 0 0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 1 0 1 0 1
  1 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 1 0 1 0 1 1 0 0 1 0 1 0
  0 0 1 1 0 0 1 0 1 0 0 1 0 0 1 1 1 0 1 1]
 [0 0 1 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 1 1 1 0 1 1 1 0 1 0 1 1 0 1 1 0
  1 0 1 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 0 1 1 1 1
  0 0 1 0 1 1 0 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0
  0 1 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 1 0 0]
 [1 1 0 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 1 1
  0 1 0 1 0 1 1 0 0 1 1 0 1 1 0 1 0 1 0 1 1 0 1 0 0 1 0 0 0 0 1 1 1 1 0 1
```

```
  0 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 0 1 1
  0 0 0 1 0 0 0 1 1 0 1 1 1 0 1 0 0 0 0 0]
 [1 0 1 0 1 1 1 1 1 1 0 0 1 0 0 1 0 0 1 0 1 0 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0
  1 1 1 1 1 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 1 0 0 0 0 1 1 1 1 0 0 0 1 0 0 0
  1 0 0 1 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 0 1 0 1 1 0 0 0 0
  0 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 1 0 1 1]
 [1 1 0 1 1 1 1 0 0 0 0 1 0 1 0 0 1 1 0 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1 1 0
  1 0 0 1 1 0 1 1 1 0 1 1 1 1 0 1 0 1 1 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0
  1 0 0 0 1 0 0 1 1 0 1 1 0 1 1 0 0 1 1 1 0 1 0 0 0 1 0 0 0 0 1 1 0 0 0 0
  0 1 0 1 0 1 1 1 1 1 0 0 1 0 1 1 1 1 1 1 1]
 [0 1 1 0 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 0 0 1 1 1 1 1 1 1 1 0 1 0 1 1 0
  0 0 0 0 1 0 0 0 1 0 1 1 0 0 1 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1 1 0 0 0 1
  0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1 0 1
  1 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 1 0 1 1]
 [0 1 1 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 0 0 1 1 0 1 1 0 1 1
  1 0 1 0 0 1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 0 0 0 0 1 0 1 0 0 0 1 1
  1 0 1 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 1 0 0 0 0 1 1 0 1 0 0 1 0 1 1 0 1
  1 0 1 0 0 1 0 1 1 1 1 1 0 1 1 0 1 0 0 0]
 [1 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 1 1 1 0 1 1 1 1 0 0 0 1 1 1 0 0 1 1
  0 1 1 1 1 0 1 1 0 1 0 1 0 1 0 1 1 1 1 1 0 0 0 1 1 0 1 0 0 0 1 1 0 1 0 0
  0 1 1 0 0 0 0 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 1 1
  0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0 0 0]]
```

5 Quellcode

```python
import numpy as np
from itertools import product
from progressbar import progressbar


# gauss-jordan-algorithmus


def gauss_jordan(matrix, result):
    # bildung der erweiterten matrix
    result_matrix = np.concatenate((matrix, result), axis=1)
    row = 0
    for column in range(matrix.shape[1]):
        # versuche kippelement auf 1 zu setzen,
        # in dem mit passender reihe XOR-t wird
        if not result_matrix[row, column]:
            for xor_row in range(row + 1, matrix.shape[0]):
                if result_matrix[xor_row, column]:
                    result_matrix[row] ^= result_matrix[xor_row]
                    break
            else:
                # falls keine andere 1 in dieser spalte,
                # mache bei nächster spalte weiter
                continue
        # XOR-e mit jeder anderen reihe, die eine 1 in der spalte hat,
        # damit kippelement einzige 1 in dieser spalte ist
        for xor_row in range(matrix.shape[0]):
            if result_matrix[xor_row, column] and xor_row != row:
                result_matrix[xor_row] ^= result_matrix[row]
        # wenn kippelement erfolgreich erzeugt wurde,
        # gehe in nächste reihe, damit nächstes kippelement
        # in nächster reihe ist. Wenn letzte Reihe erreicht ist,
        # breche ab.
        row += 1
        if row == matrix.shape[0]:
            break
    return result_matrix


# berechnet den nullraum der gegebenen matrix
def null_space(matrix):
    # bringe erweiterte matrix in spaltenstufenform (reduced column echelon form)
    rcef = gauss_jordan(matrix.T, np.identity(matrix.shape[1], dtype=bool)).T
    # trenne die erweiterte matrix wieder in ihre zwei teile (B und C)
```

```
    top_half = rcef[: matrix.shape[0]]
    bottom_half = rcef[matrix.shape[0] :]
    null_space = []
    # füge jede spalte von C zum nullraum hinzu,
    # wenn ihre zugehörige spalte in B null ist
    for i in range(top_half.shape[1]):
        if not any(top_half[:, i]):
            null_space.append(bottom_half[:, i].reshape(-1))
    return np.array(null_space)

np.set_printoptions(threshold=np.inf)
# karten einlesen
with open(input("Pfad: ")) as f:
    n_cards, n_opening_cards, n_bits = map(int, f.readline().split())
    card_strings = []
    while line := f.readline():
        card_strings.append(line.strip())
cards_bool = [[bit == "1" for bit in card] for card in card_strings]
# cards entspricht der matrix K^T
cards = np.array(cards_bool).T

# nullraum der karten (K^T) berechnen
null_space = null_space(cards)


# gebe nullvektor aus, der so viele einsen hat,
# wie es öffnungskarten + sicherungskarte gibt
null_space_int = null_space.astype(int)
for null_vector in null_space:
    if np.sum(null_vector.astype(int)) == n_opening_cards + 1:
        print("Echte Karten: ")
        print(cards.T[null_vector].astype(int))
        break
# wenn kein solcher vektor gefunden wurde, probiere
# probiere alle linearkombinationen der nullvektoren durch
else:
    print("Kein passender Basisvektor, suche Kombination...")
    for combination_factors in progressbar(
        product([False, True], repeat=null_space.shape[0]),
        max_value=2 ** null_space.shape[0],
    ):
        combination = np.zeros(null_space.shape[1], dtype=bool)
        for i, factor in enumerate(combination_factors):
            if factor:
                combination ^= null_space[i]
        if np.sum(combination.astype(int)) == n_opening_cards + 1:
            print("Echte Karten: ")
            print(cards.T[null_vector].astype(int))
            break
```

zara–zackig.py

Literatur

[1] S. Axler, *Linear Algebra Done Right*. Berlin, Heidelberg: Springer, 3rd ed., 2014.

[2] A. F. (https://math.stackexchange.com/users/54227/alfonso-fernandez), "Probability that a random binary matrix will have full column rank?." Mathematics Stack Exchange. URL:https://math.stackexchange.com/q/564699 (Version: 2013-11-12).

[3] Wikipedia contributors, "Kernel (linear algebra) — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Kernel_(linear_algebra)&oldid=1070674634, 2022. [Online; Abgerufen 18. April 2022].