

Bonusaufgabe: Zara Zackigs Zurückkehr

4.1 Lösungsidee

Wir haben N Karten v_1, \dots, v_N , welche sowohl die $K + 1$ echten Karten von Zara enthalten als auch die $N - K - 1$ zufällig hinzugefügten Karten der „Freunde“. Wir suchen nun $K + 1$ Karten daraus, sodass eine Karte (die Sicherungskarte) das exklusive Oder (XOR) der anderen K Karten ist (der Schlüsselkarten). Bei einer solchen Auswahl gilt außerdem, dass jede Karte die Sicherungskarte sein könnte, da jede Karte das XOR der jeweils anderen Karten ist. Eine solche Auswahl lässt sich auch dadurch charakterisieren, dass das XOR aller dieser Karten 0 ist. Wir müssen also $K + 1$ Karten finden, deren exklusives Oder 0 ist.

Naiver Ansatz

Es gibt insgesamt $\binom{N}{K+1}$ Möglichkeiten, aus den N Karten $K + 1$ auszuwählen. Das sind für das Beispiel aus der Aufgabenstellung (111 Karten, mit 10 Schlüsselkarten und einer Sicherungskarte) genau 473 239 787 751 081 mögliche Kombinationen. Für jede Kombination kann das XOR berechnet und mit 0 verglichen werden. Die hohe Anzahl an Kombinationen würde jedoch zu einer inakzeptablen Laufzeit führen.

Darstellung als lineares Gleichungssystem

Jede Karte kann als ein Bitvektor mit M Bits verstanden werden. Um Zaras Problem weiter in ein Problem der linearen Algebra umzuformen, definieren wir N unbekannte Variablen $s_1, \dots, s_N \in \{0, 1\}$, wobei $s_i = 1$ bedeutet, dass die Karte v_i in der Lösung ist, und $s_i = 0$, dass die Karte von den „Freunden“ stammt. Eine Belegung von s_i ist dann korrekt, wenn sie das lineare Gleichungssystem

$$v_1 s_1 \oplus v_2 s_2 \oplus \dots \oplus v_N s_N = 0,$$

wobei \oplus für das exklusive Oder steht, löst und die Nebenbedingung erfüllt, dass genau $K + 1$ der s_i gleich 1 sind und die anderen 0.

Betrachten wir ein Beispiel mit $N = 4$ Karten (Vektoren) mit jeweils $M = 6$ Bits (Länge der Vektoren). Gesucht sind $K + 1 = 3$ Karten, darunter $K = 2$ Schlüsselkarten und eine Sicherungskarte. Die Karten werden durch die folgenden Vektoren dargestellt:

$$v_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, v_3 = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, v_4 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Es entsteht folgendes Gleichungssystem:

$$v_1 s_1 \oplus v_2 s_2 \oplus v_3 s_3 \oplus v_4 s_4 = 0$$

Oder explizit:

$$\begin{aligned} 1s_1 \oplus 0s_2 \oplus 0s_3 \oplus 1s_4 &= 0 \\ 0s_1 \oplus 1s_2 \oplus 1s_3 \oplus 1s_4 &= 0 \\ 1s_1 \oplus 1s_2 \oplus 1s_3 \oplus 1s_4 &= 0 \\ 1s_1 \oplus 0s_2 \oplus 0s_3 \oplus 0s_4 &= 0 \\ 0s_1 \oplus 1s_2 \oplus 0s_3 \oplus 1s_4 &= 0 \\ 1s_1 \oplus 0s_2 \oplus 0s_3 \oplus 1s_4 &= 0 \end{aligned}$$

Dabei sollen genau 3 der s_i gleich 1, die anderen 0 sein.

Lösung mit dem Gauß-Jordan-Verfahren

Zaras Problem lässt sich so umformuliert mithilfe des Gauß-Jordan-Verfahrens¹¹ und geschickten Ausprobierens der Schlüsselmengen lösen.

Notation Die Karten können als Spaltenvektoren in eine Matrix eingetragen werden. In der ersten Zeile stehen so alle Bits an der ersten Position der Karten, in der zweiten Zeile alle Bits an der zweiten Position und so weiter. Die Bits sind somit die Koeffizienten der Matrix.

Zusammen mit der Und-Operation als Multiplikation kann das lineare Gleichungssystem auch mit einer Matrix-Vektor-Multiplikation dargestellt werden. Obiges Beispiel wird dann zu:

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Es wird also ein Vektor $S = (s_1, \dots, s_N)^T$ gesucht, sodass diese Gleichung erfüllt ist und genau $K + 1$ Karten ausgewählt werden, also $K + 1$ der s_i gleich 1 sind und die anderen gleich 0.

Gauß-Jordan-Verfahren Im Folgenden werden die Zeilen dieser Matrix mithilfe des Gauß-Jordan-Verfahrens so manipuliert, dass eine reduzierte Stufenform¹² entsteht. Hierfür wird zunächst der Nullvektor zur Matrix hinzugefügt. Dieser bleibt jedoch immer 0 und wird nur gemäß der gebräuchlichen Notation mitgeführt.

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{array} \right)$$

¹¹Siehe auch https://en.wikipedia.org/wiki/Gaussian_elimination

¹²Siehe auch https://en.wikipedia.org/wiki/Row_echelon_form#Reduced_row_echelon_form

Zunächst wird bei allen Zeilen außer der ersten versucht, die erste 1 von links zu eliminieren. Hierzu wird die erste Zeile auf alle anderen Zeilen addiert, welche ebenfalls eine führende 1 haben. Zur Erinnerung: Die Addition entspricht hier dem Exklusiv-Oder-Operator \oplus .

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ \textcolor{red}{0} & \textcolor{red}{1} & \textcolor{red}{1} & \textcolor{red}{1} & 0 \\ \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{red}{0} & 0 \\ 0 & 1 & 0 & 1 & 0 \\ \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{red}{0} & \textcolor{red}{0} & 0 \end{array} \right)$$

Das Gleiche passiert nun auch für die zweite, dritte und vierte Spalte. Sollte die i -te Zeile an der i -ten Spalte keine 1 haben, so wird sie mit einer Zeile vertauscht, welche dort eine 1 hat. Das Ergebnis ist dadurch weiterhin richtig, da nur für alle Karten gleichermaßen die Reihenfolge der Bits vertauscht wird und jede Karte weiterhin die eigenen Bits behält. Gibt es keine solche Zeile, wird selbiges in der nächsten Spalte versucht. Es entsteht folgende Matrix:

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

Hierbei sind nun Nullen unterhalb der Hauptdiagonalen entstanden. Analog wird auch für den Teil oberhalb dieser Diagonalen verfahren. Beginnend von der untersten Zeile werden nach oben Zeilen addiert, falls diese in der selben Spalte eine 1 haben wie die Zeile, welche dort eine führende 1 hat. In diesem Beispiel wird nur auf die zweite Zeile die dritte addiert. In der ersten und zweiten Zeile sind zwar ebenfalls 1en in der gleichen Spalte, allerdings ist keine von beiden eine führende 1. So entsteht folgende Matrix:

$$\left(\begin{array}{cccc|c} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & \textcolor{red}{0} & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

Bestimmen der Teillösungen Nachdem einige Teile der Matrix auf Nullen reduziert werden konnten, wird abgezählt, wie viele Stufen die Stufenform hat. Bei unserem Beispiel sind dies drei Stufen, hier rot markiert:

$$\left(\begin{array}{cccc|c} \textcolor{red}{1} & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ \textcolor{red}{0} & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} \right)$$

Da $K = 2$ ist, wird eine Lösung mit 3 Karten gesucht. Für alle Spalten (Karten) rechts des markierten Teils muss nun eine Auswahl getroffen werden, welche Spalten in der Lösung vorkommen und welche nicht. Für die Spalten links davon gibt es jeweils eine eindeutige Auswahl (die aus der Stufenform abgelesen werden kann), die das Gleichungssystem löst, jedoch nicht unbedingt die Nebenbedingung erfüllt. Für jede Auswahl muss also geprüft werden, ob die Nebenbedingung erfüllt ist.

In diesem Beispiel gibt es nur eine freie Spalte (die 4. Spalte), für welche die beiden Möglichkeiten (in der Lösung oder nicht) getestet werden müssen.

1. Betrachten wir zunächst den Fall, dass die 4. Karte nicht in der Lösung ist ($s_4 = 0$) und somit alle Werte der 4. Spalte mit 0 multipliziert werden. Nun steht in den ersten drei Zeilen, dass s_1, s_2 und s_3 jeweils 0 sind, wie im Gleichungssystem zu erkennen ist:

$$\begin{aligned} 1s_1 \oplus 0s_2 \oplus 0s_3 \oplus 1 \cdot 0 &= 0 \\ 0s_1 \oplus 1s_2 \oplus 0s_3 \oplus 1 \cdot 0 &= 0 \\ 0s_1 \oplus 0s_2 \oplus 1s_3 \oplus 0 \cdot 0 &= 0 \end{aligned}$$

Also ist die Gesamtauswahl leer und die Bedingung, dass genau $K + 1 = 3$ Karten ausgewählt werden müssen, nicht erfüllt.

2. Wird jedoch angenommen, dass die 4. Karte in der Lösung ist ($s_4 = 1$), so entstehen folgende Gleichungen:

$$\begin{aligned} 1s_1 \oplus 0s_2 \oplus 0s_3 \oplus 1 \cdot 1 &= 0 \\ 0s_1 \oplus 1s_2 \oplus 0s_3 \oplus 1 \cdot 1 &= 0 \\ 0s_1 \oplus 0s_2 \oplus 1s_3 \oplus 0 \cdot 1 &= 0 \end{aligned}$$

Die konstanten Terme werden auf die rechte Seite gebracht:

$$\begin{aligned} 1s_1 \oplus 0s_2 \oplus 0s_3 &= 1 \\ 0s_1 \oplus 1s_2 \oplus 0s_3 &= 1 \\ 0s_1 \oplus 0s_2 \oplus 1s_3 &= 0 \end{aligned}$$

Es gilt also $s_1 = s_2 = 1$ und somit sind die Karten v_1 und v_2 in der Lösung, zusammen mit unserer Vorauswahl v_4 . In der Tat erfüllt dies die Nebenbedingung, dass die Auswahl aus drei Karten bestehen muss. Die Lösung lautet also v_1, v_2, v_4 .

Sofern es Spalten rechts der Stufenform gibt (bei den BWINF-Beispielen in der Regel $N - M$ Karten, bzw. 0, wenn $N < M$), muss jede Auswahl daraus auf Gültigkeit getestet werden, indem gezählt wird, wie viele der Einträge den Wert 1 haben. Für die gebundenen Variablen lassen sich wie oben bereits erwähnt aus der Stufenformel explizite Terme in Abhängigkeit der freien Variablen ablesen, in die jeweils nur eingesetzt werden muss.

Es müssen natürlich nur Auswahlen der freien Variablen geprüft werden, bei denen höchstens $K + 1$ der Einträge 1 sind.

Beim Bestimmen der Lösungen bietet es sich an, die Anzahl der gewählten freien Spalten erst zu erhöhen, nachdem alle Möglichkeiten mit einer Anzahl getestet wurden. Mit anderen Worten: Zuerst testet man alle Kombinationen mit null Spalten, dann alle Kombinationen mit einer Spalte, dann mit zwei Spalten und so weiter. Dies hängt damit zusammen, dass die Wahrscheinlichkeit gering ist, dass alle $K + 1$ gesuchten Karten in den letzten $N - M$ Spalten vorkommen,

wenn wir davon ausgehen, dass die „Freunde“ gut gemischt haben und die Verteilung der Karten gleichverteilt zufällig ist.

Eindeutigkeit

Um zu testen, ob die gefundene Lösung die einzige Lösung ist, könnten alle Möglichkeiten getestet werden. Da jedoch in der Aufgabenstellung beschrieben wurde, dass die hineingemischten Karten zufällig generiert wurden, ist die Wahrscheinlichkeit für weitere Lösungen sehr gering: Die Exklusiv-Oder-Summe jeder der $\binom{N}{K+1} - 1$ nicht richtigen Teilmengen ist unabhängig gleichverteilt zufällig (unter den 2^M Möglichkeiten), und daher ist die Wahrscheinlichkeit einer zufälligen Lösung:

$$P(N, M, K) = 1 - \left(1 - \frac{1}{2^M}\right)^{\binom{N}{K+1}-1}.$$

Für die Beispiele auf der BWINF-Website ist diese Abschätzung immer kleiner als 10^{-4} .

Implementierung

Für die Implementierung eignen sich Bit-Arrays. Ein *Bit-Array* ist ein Array von Bits, das auf einer Maschine mit 64-Bit-Integern als (kürzeres) Array von Integern dargestellt werden kann, deren 64 Bits jeweils bis zu 64 Bits des Bit-Arrays entsprechen.

Nachdem die Karten eingelesen wurden, werden Bit-Arrays verwendet, um die Zeilen der Matrix darzustellen, und die Werte der Karten als Spaltenvektoren in die Matrix eingetragen. Darauf folgt das Gauß-Jordan-Verfahren, angewendet auf die Zeilen. Die dabei nötigen Zeilenoperationen (hier: Additionen von Zeilen) werden, da es sich um Operationen von Bit-Arrays handelt, um einen konstanten Faktor, etwa 64, schneller durchgeführt.

Nachdem die Anzahl der Stufen festgestellt wurde, werden die noch freien Spaltenvektoren wiederum in Bit-Arrays umgewandelt und daraufhin mit diesen Arrays das Bestimmen der restlichen Schlüssel durchgeführt. Auch die dafür nötigen Berechnungen werden dadurch um einen konstanten Faktor schneller.

Hierfür lohnt es sich, eine schnelle Methode zu entwickeln, die Anzahl der Einsen in einem Integer zu zählen. Hierfür kann neben der naiven Methode auch Brian Kernighans Algorithmus¹³, Bitmanipulation¹⁴ oder die Assembly-Instruktion `popcnt`¹⁵ verwendet werden.

Laufzeitanalyse

Bei N verschiedenen Karten der Länge M das Gauß-Jordan-Verfahren anzuwenden, hat einen Aufwand von

$$T = \mathcal{O}(NM^2),$$

da wir jede der M Zeilen auf höchstens M verschiedene Zeilen addieren müssen und diese Operation für je N Bits ausführen. Die Laufzeit der Bestimmung der Teillösungen ist abhängig davon, wie viele Stufen bei dem Gauß-Jordan-Verfahren entstanden sind. Gilt $N < M$, gibt es

¹³Siehe <https://graphics.stanford.edu/seander/bithacks.html#CountBitsSetKernighan>

¹⁴Siehe <http://graphics.stanford.edu/~seander/bithacks.html#CountBitsSetParallel>

¹⁵Siehe <https://www.felixcloutier.com/x86/popcnt>

also weniger Karten, als jede Karte Bits hat, wie bei dem Beispiel aus der Aufgabenstellung, so werden meist N Stufen entstehen und der Teil, der im Weiteren betrachtet werden muss, ist gering (bzw. nicht vorhanden). Sollte jedoch $N > M$ gelten, es also mehr Karten geben, als jede Karte Bits hat, wie bei `stapel3.txt`, `stapel4.txt` und `stapel5.txt`, so muss bei mindestens $N - M$ Spalten geprüft werden, ob sie in der Lösung sind oder nicht. Da einige Teillösungen direkt ausgeschlossen werden, und wenn wir davon ausgehen, dass $N - M$ Spalten rechts von der Stufenform übrig bleiben, so müssen trotzdem noch bis zu

$$\sum_{i=0}^{K+1} \binom{N-M}{i}$$

Auswahlen an Spalten überprüft werden und für jede davon die restlichen Variablen s_i berechnet werden. Im Erwartungswert enthält die richtige Auswahl jedoch nur $\frac{K+1}{N}(N-M)$ der $N - M$ Karten, sodass die tatsächliche Anzahl an überprüften Auswahlen deutlich geringer ist.

4.2 Auswahl des korrekten Schlüssels

Nachdem Zara ihre Schlüsselkarten $v_{j_1}, v_{j_2}, \dots, v_{j_{K+1}}$ gefunden hat, kann sie diese erneut aufsteigend sortieren (wir nehmen an, dass dies bereits die Reihenfolge nach Sortierung ist). Möchte sie nun die i -te Tür öffnen, kann sie zunächst die Karte v_{j_i} testen. Sollte diese Karte nicht die Tür öffnen, kann dies nur daran liegen, dass der Sicherungsschlüssel vor dem gesuchten Schlüssel eingeordnet wurde. Also wurde der gesuchte Schlüssel um eins verschoben, woraus folgt, dass die Karte $v_{j_{i+1}}$ die richtige ist.

Zara wird so tatsächlich nicht mehr als *einen* Fehlversuch benötigen, um das Haus aufzusperren.

4.3 Weiterer Lösungsansatz (Teile und Herrsche)

Das Problem lässt sich auch mit dem folgenden Teile-und-Herrsche-Algorithmus¹⁶ angehen.

Zunächst speichert man für alle Teilmengen der ersten $\lfloor (K+1)/2 \rfloor$ Karten ihr Exklusiv-Oder zusammen mit der Teilmenge in einer nach dem Exklusiv-Oder sortierten Datenstruktur. Diese kann beispielsweise als Hashtabelle realisiert werden. Daraufhin wird für alle Teilmengen der restlichen $\lceil (K+1)/2 \rceil$ Karten getestet, ob ihr Exklusiv-Oder in der Datenstruktur ist oder nicht. Ein Treffer entspricht einer Lösung, nämlich der Vereinigung der beiden Teilmengen von Karten. Dieser Algorithmus hat einen Zeitaufwand von

$$T = \mathcal{O}\left(M \cdot \binom{N}{\lceil (K+1)/2 \rceil} \cdot \log \lceil (K+1)/2 \rceil\right).$$

Der logarithmische Faktor ist für das Suchen von Elementen in der Datenstruktur nötig. Durch diesen Trick der Aufteilung der Karten in zwei Teile ergibt sich also eine deutlich bessere Laufzeit im Vergleich zu der der naiven Lösung von $\mathcal{O}\left(M \cdot \binom{N}{K+1}\right)$.

Mit diesem Algorithmus lassen sich alle Beispiele außer `stapel3.txt` und `stapel4.txt` in unter einer Stunde lösen. Wirklich geeignet ist er aber vor allem bei `stapel5.txt`, worauf er,

¹⁶Siehe auch https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm

da $N - M = 136$ sehr groß, aber dafür $K = 4$ sehr klein ist, eine deutlich bessere Laufzeit als der das Gauß-Jordan-Verfahren benutzende Lösungsansatz hat. Bei dieser Aufgabe sind also grundsätzlich verschiedene Ansätze für in verschiedenen Dimensionen große Eingaben jeweils besser oder schlechter geeignet. Insbesondere kann nicht pauschal davon gesprochen werden, dass einer dieser beiden Ansätze die *bessere* Laufzeit habe.

4.4 Beispiele

Im folgenden Abschnitt werden die (mit sehr hoher Wahrscheinlichkeit eindeutigen) Lösungen für die von BWINF vorgegebenen Beispiele vorgestellt. Bei jeder Lösung wird eine Tabelle angegeben mit den Indizes (beginnend mit 0, nach der Reihenfolge in der Datei) der Karten zusammen mit den ersten 32 Bits der Codewörter. Die Karten sind für Zara aufsteigend sortiert.

stapel0.txt

$$N = 20, M = 32, K = 4$$

Laufzeit < 1 ms

Index				
2	00111101	01011100	01101001	10011001
9	10101100	11111101	10101000	11100000
11	10111000	01100111	00001010	10111110
5	11010111	11101011	11011011	11110000
3	11111110	00101101	00010000	00110111

stapel1.txt

$$N = 20, M = 32, K = 8$$

Laufzeit < 1 ms

Index				
15	00010001	11010011	00011111	01100100
1	00100000	11110011	11101111	01111100
11	00100011	10011101	10101110	11100011
4	00110100	00101010	01000011	11010010
7	00110110	00011010	11010111	11111010
14	11000111	11101011	01000001	01110100
2	11010011	01011011	01010011	01010111
6	11110011	10101100	10010000	10111110
9	11110111	10010001	01001000	01001110

stapel2.txt

$$N = 111, M = 128, K = 10$$

Laufzeit ≈ 1 ms

Index						
57	00101000	01100001	00101110	11101011	...	
20	00101011	11100010	10110101	10111100	...	
89	01101001	00101100	01010011	11111101	...	
9	01101011	10100011	01110100	01100001	...	
95	01110110	01111000	11100111	10001101	...	
53	10000000	00010010	01100110	01000110	...	
26	10101011	00000110	11000001	01111111	...	
76	10101111	11001001	00101001	11101100	...	
71	11000011	00010011	01110001	01100100	...	
78	11011110	00010100	11011111	00110000	...	
99	11101110	10101110	01111011	11000111	...	

stapel3.txt

$$N = 161, M = 128, K = 10$$

Laufzeit ≈ 2 ms

Index						
133	00100000	11100111	00011010	10001111	...	
43	01010000	10110111	00111100	01110011	...	
120	01110001	11111010	10000100	11100011	...	
127	01110110	10011100	10000110	11100100	...	
98	01111101	10001010	11001100	11101011	...	
71	10110000	00100110	01101101	01000100	...	
23	10110111	01001011	11101100	11000101	...	
26	10111000	10111110	00101111	10101010	...	
110	10111111	01010100	11000001	01101100	...	
144	11000001	01100100	01101001	11011111	...	
34	11001011	01011111	11101110	10001000	...	

stapel4.txt

$$N = 181, M = 128, K = 10$$

Laufzeit ≈ 8 ms

Index						
76	00000111	01101001	01011011	10001111	...	
23	00101100	00011110	11110000	10000001	...	
91	00101100	00111000	11100111	10000100	...	
83	00110110	01011100	10010011	11001111	...	
95	01000011	00100110	10110011	11011101	...	
116	10000001	00010111	00110101	00011000	...	
159	10001100	00101100	11010010	11000110	...	
154	10101010	00001111	11110011	11011110	...	
146	11000101	11000100	10000010	11101000	...	
0	11100010	00000011	11010011	11111001	...	
160	11110010	11000110	00101001	10001001	...	

stapel5.txt

$$N = 200, M = 64, K = 4$$

Laufzeit $\approx 865\text{ ms}$

Index						
167	01011111	11000111	00000010	11111000	...	
70	10000100	11101010	00111110	01001101	...	
185	10100001	10101100	10111011	10011000	...	
163	10101110	11001100	10011000	11001100	...	
77	11010100	01001101	00011111	11100001	...	

4.5 Bewertungskriterien

Die aufgabenspezifischen Bewertungskriterien werden hier erläutert.

1. Lösungsweg

- (1) *Problem adäquat modelliert:* Die Karten müssen vollständig verwendet und das bitweise Exklusiv-Oder korrekt berechnet werden. Zudem muss in Betracht gezogen werden, dass es mehrere Lösungen geben kann.
- (2) *Laufzeit des Verfahrens in Ordnung:* Jeder Algorithmus mit einer Laufzeit von $\mathcal{O}\left(\binom{N}{K+1}\right)$ oder schlechter ist als ineffizient zu betrachten und mit Punktabzug zu bewerten. Algorithmen, welche die Laufzeit stark genug einschränken, sollten hier keinen Abzug bekommen.
Praktischer gesprochen: Es gibt viele Verfahren, welche die kleineren Beispiele lösen können. Sollte ein Verfahren alle Beispiele lösen, ist es als sehr gut einzuschätzen. Falls ein Verfahren das Beispiel aus der Aufgabenstellung löst (und wohl auch die kleineren Beispiele), gibt es keine Punktabzüge. Falls auch die herausfordernden Beispiele (stapel3.txt bis stape15.txt) gelöst werden, so können bis zu zwei Bonuspunkte vergeben werden: 1 Punkt für mindestens zwei dieser Beispiele, 2 Punkte für alle drei. Ein besonderer Fall ist der geschilderte Ansatz nach dem Prinzip „teile und herrsche“. Wird er allein verwendet, können die Beispiele 3 und 4 nicht gelöst werden und Beispiel 2 nur mit Mühe; dann werden Punkte abgezogen. Gleichzeitig ist er aber wichtig für Beispiel 5, so dass es ohne ihn möglicherweise nicht die vollen Bonuspunkte geben kann.
- (3) *Verfahren nicht unnötig ineffizient:* Es sollte eine Datenstruktur verwendet werden, die für den gewählten Algorithmus geeignet ist. So ist zum Beispiel die Verwendung von Bitarrays oder Bitvektoren notwendig für eine effiziente Implementierung, wenn diese auf dem Gauß-Jordan-Verfahren basiert.
- (4) *Verfahren mit korrekten Ergebnissen:* Das Ergebnis ist immer eine Auswahl an $K + 1$ Karten, die alle in der Ursprungsmenge vorkommen. Die Exklusiv-Oder-Summe über die finale Auswahl der Schlüssel sollte den Nullvektor ergeben.
- (5) *Verfahren funktioniert uneingeschränkt:* Das Verfahren kann Problemstellungen für alle möglichen N, M und K bearbeiten. Es dürfen keine Bedingungen an die Eingabeparameter gesetzt werden, wie zum Beispiel, dass N kleiner als M ist.

- (6) *Verfahren zur Schlüsselauswahl angegeben:* Auch Aufgabenteil b) muss beantwortet werden. Dazu gehört die Angabe eines Verfahrens zur Auswahl des korrekten Schlüssels aus den $K + 1$ gefundenen Schlüssel- und Sicherungskarten.

2. Theoretische Analyse

- (1) *Verfahren / Qualität insgesamt gut begründet:* Das Verfahren und insbesondere seine Korrektheit müssen gut nachvollziehbar begründet werden. Bei der Verwendung bereits bekannter Algorithmen müssen diese erklärt oder referenziert werden; aus der Dokumentation muss klar werden, dass verstanden wurde, was die „fremden“ Verfahren tun und warum sie zur Lösung nützlich sind.
- (2) *Gute Überlegungen zur Laufzeit des Verfahrens:* Die Laufzeit des Verfahrens lässt sich recht einfach erkennen und sollte auch korrekt in der Dokumentation vorkommen. Die Laufzeit muss nicht zwingend formal, aber nachvollziehbar und korrekt charakterisiert werden.

3. Dokumentation

- (3) *Vorgegebene Beispiele dokumentiert:* Alle Beispiele der BWINF-Website sollten dokumentiert und nachvollziehbar sein. Sollte ein Beispiel nicht gelöst werden können, so sollte dies erwähnt werden.
- (5) *Ergebnisse nachvollziehbar dargestellt:* Die Ergebnisse zu den Beispielen sollten die Bits der Schlüsselkarten und der Sicherungskarte jeweils in einer Zeile gelistet haben. Es muss entweder der ursprüngliche Index der ausgewählten Karten mit angegeben werden oder diese zumindest sortiert abgebildet werden.