Aufgabe 2: Spießgesellen

Teilnahme-ID: 55908

Jonathan Busch

19. April 2021

Inhaltsverzeichnis

1 Annahmen

- $falsch = 0$ und $wahr = 1$, also z. B. $5 \in \mathbb{N} = 1$ und $-5 \in \mathbb{N} = 0$.
- Donald hat sich bei keiner Beobachtung geirrt. Diese Annahme vereinfacht einige Überlegungen wesentlich.

2 Lösungsidee

2.1 Teilaufgabe a)

In einer Tabelle kann man nach jeder Beobachtung festhalten, in welchen Schüsseln sich ein Obst noch befinden könnte. Betrachtet man die erste Beobachtung, Mickys Obstspieß, erhält man folgende Aussagen:

1. Die Obstsorten Apfel, Banane und Brombeere befinden sich in den Schüsseln 1, 4 und 5.
2. Die anderen Obstsorten können sich nicht in den Schüsseln 1, 4 und 5 befinden, d. h. die Obstsorten Erdbeere, Pflaume und Weintraube befinden sich in den Schüsseln 2, 3 und 6.

Obst	Menge der Schüsseln
Apfel	$\{1, 4, 5\}$
Banane	$\{1, 4, 5\}$
Brombeere	$\{1, 4, 5\}$
Erdbeere	$\{2, 3, 6\}$
Pflaume	$\{2, 3, 6\}$
Weintraube	$\{2, 3, 6\}$

Zieht man nun Minnies Obstspieß hinzu, lauten die Aussagen wie folgt:

1. Die Obstsorten Banane, Pflaume und Weintraube befinden sich in den Schüsseln 3, 5 und 6;
2. Die Obstsorten Apfel, Brombeere und Erdbeere befinden sich in den Schüsseln 1, 2 und 4.

Damit gelangt man zur folgenden neuen Tabelle, indem man die Aussagen logisch kombiniert. Letztendlich bildet man dabei die Schnittmenge zwischen der Menge der bisher möglichen Schüsseln für eine Obstsorte und der Menge der Schüsseln, die die aktuelle Beobachtung dem Obst zuschreibt.

Obst	Menge der Schüsseln
Apfel	$\{1, 4\} = \{1, 4, 5\} \cap \{1, 2, 4\}$
Banane	$\{5\} = \{1, 4, 5\} \cap \{3, 5, 6\}$
Brombeere	$\{1, 4\} = \{1, 4, 5\} \cap \{1, 2, 4\}$
Erdbeere	$\{2\} = \{2, 3, 6\} \cap \{1, 2, 4\}$
Pflaume	$\{3, 6\} = \{2, 3, 6\} \cap \{3, 5, 6\}$
Weintraube	$\{3, 6\} = \{2, 3, 6\} \cap \{3, 5, 6\}$

Nach weiteren Beobachtungen kommt man schließlich zu folgender Tabelle:

Obst	Menge der Schüsseln
Apfel	$\{1, 4\}$
Banane	$\{5\}$
Brombeere	$\{1, 4\}$
Erdbeere	$\{2\}$
Pflaume	$\{6\}$
Weintraube	$\{3\}$

Sowohl die Schüsseln der Äpfel als auch der Brombeeren sind also auch nach der letzten Beobachtung noch nicht eindeutig bestimmt. Dieser Kenntnisstand reicht für Donald trotzdem aus, da er ohnehin beide Obstsorten essen möchte. Er muss sich also aus den Schüsseln 1, 3 und 4 bedienen.

2.2 Allgemeine Problemstellung

Gegeben sind eine Menge von Obstsorten O und eine Menge von Schüsseln $S = \{1, 2, ..., |O|\}$, in denen sich die Obstsorten befinden. Befindet sich ein Obst $x \in O$ in einer Schüssel $y \in S$, soll dafür $s(x) = y$ und $o(y) = x$ notiert werden. Außerdem ist eine Menge B von Beobachtungen der Form $(P \subset O, T \subset S)$ gegeben, die jeweils angeben, dass $\forall p \in P$: $s(p) \in T \ \wedge \ \forall t \in T$: $o(t) \in P$.

Das Ziel ist, für eine Menge von Wunschobstsorten $W \subset O$ eine Menge von Schüsseln $X \subset S$ zu bestimmen, sodass $X = \{s(x) | x \in W\}$.

2.3 Teilaufgabe b)

Wie aus Teilaufgabe a) bekannt, enthalten die Beobachtungen jeweils logische Aussagen darüber, welche Obstsorten sich in bestimmten Schüsseln befinden bzw. nicht befinden könnten.

Die wesentliche Idee ist, in einer Obstmatrix M festzuhalten, ob sich ein Obst i in einer Schüssel j befinden könnte. Ist das der Fall, soll $M_{i,j} = 1$ gelten, andernfalls $M_{i,j} = 0$. Solche Matrizen kann man sowohl für einzelne Aussagen als auch für den gesamten aktuellen Kenntnisstand verwenden. Die logische Kombination von zwei Aussagenmatrizen erhält man, indem man das Hadamard-Produkt bildet (indem man die Matrizen komponentenweise multipliziert, was hier der logischen Konjunktion entspricht). Es gilt also für die Kombination E der Aussagenmatrizen C und D: $E = C \circ D \iff E_{i,j} = C_{i,j} \wedge D_{i,j}$.

Aus praktischen Gründen wird die Menge der Beobachtungen in einer neuen Form (als Matrix) definiert: $B' = \{b'_{i,j} = (i \in P \iff j \in T) \mid (P, T) \in B\}$. Aus einer Beobachtung (P, T) wird also eine Matrix b' generiert, für die der Eintrag $b'_{i,j}$ genau dann 1 ist, wenn die Aussagen $i \in T$ und $j \in P$ äquivalent sind, d. h. entweder wenn das Obst i auf dem Spieß vorkam und die beobachtete Person die Schüssel j aufgesucht hat, oder wenn keine der beiden Aussagen zutrifft.

Das Problem lässt sich schließlich lösen, indem man $G = \prod_{b \in B'} b$ als Hadamard-Produkt berechnet. Die Reihenfolge spielt dabei keine Rolle, da das Hadamard-Produkt kommutativ ist.

Ergebnis dieser Multiplikation ist also der Gesamtkenntnisstand G in Form einer Matrix. In dieser Matrix lässt sich auch recht einfach für ein Obst i ablesen, in welchen Schüsseln es sich befinden könnte: $s(i) \in \{j \mid G_{i,j} = 1\}$. Die Menge X der Schüsseln, die Donald aufsuchen muss, ist also, sofern sie eindeutig bestimmt ist, $X = x_{w_1} \cup x_{w_2} \cup ... \cup x_{w_{|W|}}$ für $x_{w_i} = \{j \mid G_{w_i,j} = 1\}$ und $w_i \in W$. Die Menge ist genau dann nicht eindeutig betimmt, wenn $|X| > |W|$. Wenn $|X| = |W|$, gibt es nämlich pro Obstsorte in W genau eine Schüssel in X. Wenn $|X| < |W|$, dann gibt es mindestens eine Obstsorte, die in keiner Schüssel sein kann. Dieser Fall kann nicht vorkommen, wenn Donald sich nicht geirrt hat (Annahme). Sonst ($|X| > |W|$) gibt es mindestens eine Schüssel in X, die eine Obstsorte enthält, das nicht in W ist (Dieser Zustand sollte vermieden werden, da Donald beim Aufsuchen dieser Schüssel zugeben müsste, dass er das Obst nicht essen will, was vermutlich peinlich wäre...). In diesem Fall werden die kritischen Obstsorten in W bestimmt, um noch eine „möglichst informative Meldung“ auszugeben, die aus Informationen bestehen soll, welche Schüsseln und Obstsorten Donald noch besonders beobachten muss, um die Menge schließlich eindeutig zu bestimmen.

Ob eine Obstsorte kritisch ist, soll nun durch die Funktion $kritisch(p) = \exists y \in (O \backslash W) : y \in x_p$ angegeben werden. Die Menge der kritischen Obstsorten ist zunächst wie folgt definiert: $O_{Kritisch} = \{w \in W \mid kritisch(w)\}$, also $O_{Kritisch} \subset W$. Daraus kann man die Menge der zu beobachtenden Schüsseln ableiten: $S_{beob} = x_{k_1} \cup x_{k_2} \cup ... \cup x_{k_{|O_{Kritisch}|}}$, also die Menge aller Schüsseln, in denen sich kritische Obstsorten befinden könnten. Dementsprechend sind die zu beobachtenden Obstsorten $O_{beob} = \{p \in O \mid (|x_p \cap S_{beob}|) > 0\}$, also die Menge aller Obstsorten, die sich in zu beobachtenden Schüsseln befinden könnten.

3 Korrektheit

Letztendlich wird in den Verfahren für jedes Paar (a, b) aus dem Obst a und der Schüssel b berechnet, ob sich a in b befinden könnte. Jede Beobachtung macht für jedes Paar genau eine Aussage (ob das möglich ist (1), oder nicht (0)). Wenn alle Beobachtungen auf diese Weise zustimmen, dass sich a in b befinden könnte, dann kann man diese Möglichkeit auch insgesamt nicht ausschließen. Wird die Möglichkeit jedoch durch mindestens eine Beobachtung eindeutig widerlegt, so kann man die Möglichkeit insgesamt ausschließen. Die logische Verknüpfungsoperation für Aussagen ist also die logische Konjunktion, die numerisch ($wahr = 1 \wedge falsch = 0$) dem Produkt entspricht. Da das Hadamard-Produkt nichts anderes als das komponentenweise numerische Produkt der Einträge einer Matrix ist, werden darüber alle logischen Konjunktionen parallel berechnet. Das Verfahren liefert also tatsächlich korrekte Ergebnisse.

4 Laufzeitanalyse

Die Laufzeit ist abhängig von der Anzahl der Obstsorten n und der Anzahl der Beobachtungen b (Bezeichnungen weichen im Programm ab).

Die Berechnung einer Beobachtungsmatrix aus den Mengen (P, T) braucht $\mathcal{O}(n)$ Zeit. Betrachtet man eine Beobachtungsmatrix als Adjazenzmatrix eines bipartiten Graphen, so wird

offensichtlich, dass der Graph aus zwei bipartit vollständigen Teilgraphen besteht: (P, T) und $(O \setminus P, S \setminus T)$. Dadurch wird deutlich, dass, betrachtet man die Beobachtungsmatrix nun zeilenweise, sie nur zwei unterschiedliche Zeilen haben kann: eine Zeile mit $x_i = i \in T$ und eine Zeile mit $x_i = x \in (S \setminus T)$. Beide Zeilen lassen sich jeweils in $\mathcal{O}(n)$ berechnen, und die Zuweisung der Daten zu den n Zeilen kann (mit Zeigern, um nicht zu kopieren[1]) in $\mathcal{O}(n)$ durchgeführt werden.

Die Berechnung der komponentenweisen logischen Konjunktion zweier Matrizen braucht $\mathcal{O}(n^2)$ Berechnungsschritte. Bei b Beobachtungsmatrizen läuft die Berechnung also in $\mathcal{O}(n^2 b)$.

5 Umsetzung

Das Programm wird in Java implementiert.

Da die Anzahl der Obstsorten durch 26 beschränkt ist und $26 \leq 32$, können die Matrizen zeilenweise als `int[]` gespeichert werden, wobei der Eintrag $_{i,j}$ durch das j-te Bit des Integers an Arrayindex i repräsentiert wird. Diese Darstellung hat den Vorteil, dass die logische Kombination mehrerer Aussagen durch den bitweisen `&`-Operator berechnet werden kann. Außerdem repräsentiert `m[i]` im Array die Menge der Schüsseln, in denen sich das Obst i befinden könnte. Die Menge X, die Vereinigungsmenge aller Schüsseln, in denen sich die Wunschobstsorten W befinden können, kann man also mit dem bitweisen `|`-Operator berechnen, ebenso wie die Menge der zu beobachtenden Schüsseln, die ebenfalls als Vereinigungsmenge definiert ist.

Die Laufzeit dieses Umsetzungsansatzes kann man nun auch als $\mathcal{O}(b\ n\ \lceil \frac{n}{32} \rceil)$ schreiben. Da $\lceil \frac{n}{32} \rceil = 1$ für $n \leq 32$, liegt die Laufzeit deshalb praktisch in $\mathcal{O}(n\ b)$.

Da dieser Umsetzungsansatz auf 32 Obstsorten beschränkt ist, wurde ein zweites Programm `A2MainExt` geschrieben, das statt mit `int`s mit `BitSet`s arbeitet und daher viel mehr Obstsorten verträgt, aber bei der gleichen Anzahl an Obstsorten und Beobachtungen wahrscheinlich[2] langsamer ist. Hier gilt die Einschränkung der Laufzeitklasse nicht, da $n \leq 32$ nicht mehr gilt.

6 Beispiele

Als erstes das Beispiel aus der Aufgabenstellung (in den Dateien ergänzt als `spiesse8.txt`):

```
--- Befehl ---
$ java A2Main # Beispiel 8
```

[1] In `A2Main` wird darauf keine Rücksicht genommen, da in dieser Implementation `int`s als Bitmasken verwendet werden, die sich in $\mathcal{O}(1)$ kopieren lassen. Die Verwendung von Zeigern wird erst in `A2MainExt` für mehr als 32 Obstsorten relevant, in der `BitSet`s als Bitmasken verwendet werden müssen. Dabei muss allerdings auch beachtet werden, dass die Ausgangsobstmatrix für die Berechnung mit einer vollbesetzten Matrix initialisiert wird, in der jede Zeile ein eigenes `BitSet`-Objekt hat.

[2] Das wurde nicht getestet. Da Datenobjekte (wie `BitSet`) in Java aber normalerweise langsamer sind als primitive Datentypen, liegt diese Vermutung nahe. Der Unterschied dürfte allerdings kaum messbar sein, da das erste Programm, das mit `int`-Werten arbeitet, auf 32 Obstsorten beschränkt ist und seine absolute Laufzeit in der Praxis daher grundsätzlich vernachlässigbar ist.

```
--- INPUT  ---
6
Apfel Brombeere Weintraube
4
1 4 5
Apfel Banane Brombeere
3 5 6
Banane Pflaume Weintraube
1 2 4
Apfel Brombeere Erdbeere
2 6
Erdbeere Pflaume

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Apfel: [1, 4]
Brombeere: [1, 4]
Weintraube: [3]
Banane: [5]
Pflaume: [6]
Erdbeere: [2]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 3, 4]
```

Dann die Beispiele von der BwInf-Webseite:

```
--- Befehl ---
$ java A2Main # Beispiel 1

--- INPUT  ---
10
Clementine Erdbeere Grapefruit Himbeere Johannisbeere
4
6 10 3
Banane Feige Ingwer
2 1 9 8 4
Apfel Clementine Dattel Erdbeere Himbeere
5 8 10 4 2
Apfel Erdbeere Feige Himbeere Johannisbeere
6 4 2 5 9
Dattel Erdbeere Himbeere Ingwer Johannisbeere

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Clementine: [1]
Erdbeere: [2, 4]
Grapefruit: [7]
Himbeere: [2, 4]
```

```
Johannisbeere: [5]
Banane: [3]
Feige: [10]
Ingwer: [6]
Apfel: [8]
Dattel: [9]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 2, 4, 5, 7]

--- Befehl ---
$ java A2Main # Beispiel 2

--- INPUT  ---
12
Apfel Banane Clementine Himbeere Kiwi Litschi
5
2 8 6 9 1 7 4
Apfel Dattel Erdbeere Feige Johannisbeere Kiwi Litschi
11 1 12 8 4 5 10
Apfel Banane Clementine Erdbeere Himbeere Ingwer Johannisbeere
5 2 9 8 11 3 10
Banane Clementine Dattel Erdbeere Feige Grapefruit Himbeere
4 6 12 1 2 9 3
Apfel Dattel Feige Grapefruit Ingwer Johannisbeere Kiwi
3 4 7 11 5 10
Banane Clementine Grapefruit Himbeere Johannisbeere Litschi

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Apfel: [1]
Banane: [5, 10, 11]
Clementine: [5, 10, 11]
Himbeere: [5, 10, 11]
Kiwi: [6]
Litschi: [7]
Dattel: [2, 9]
Erdbeere: [8]
Feige: [2, 9]
Johannisbeere: [4]
Ingwer: [12]
Grapefruit: [3]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 5, 6, 7, 10, 11]

--- Befehl ---
$ java A2Main # Beispiel 3

--- INPUT  ---
```

```
15
Clementine Erdbeere Feige Himbeere Ingwer Kiwi Litschi
4
4 7 9 10 12 3 2 11
Feige Grapefruit Ingwer Johannisbeere Kiwi Litschi Nektarine Orange
10 3 4 7 2 5 11 13
Clementine Dattel Feige Grapefruit Ingwer Litschi Nektarine Orange
4 12 9 5 3 14 1 6
Apfel Banane Clementine Himbeere Johannisbeere Kiwi Nektarine Orange
1 3 11 8 2 9 13
Dattel Erdbeere Grapefruit Himbeere Johannisbeere Litschi Orange

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Clementine: [5]
Erdbeere: [8]
Feige: [7, 10]
Himbeere: [1]
Ingwer: [7, 10]
Kiwi: [12]
Litschi: [2, 11]
Grapefruit: [2, 11]
Johannisbeere: [9]
Nektarine: [4]
Orange: [3]
Dattel: [13]
Apfel: [6, 14]
Banane: [6, 14]
null: [15]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 2, 5, 7, 8, 10, 11, 12]
Achtung! Schüsseln könnten auch andere Obstsorten beinhalten! Kritisch sind
    folgende Gruppen:
[Litschi, Grapefruit] in [2, 11]
Diese Schüsseln und Obstsorten sollten genauer beobachtet werden!

--- Befehl ---
$ java A2Main # Beispiel 4

--- INPUT  ---
17
Apfel Feige Grapefruit Ingwer Kiwi Nektarine Orange Pflaume
6
16 7 9 1 2 11 6 15 8
Apfel Clementine Erdbeere Grapefruit Himbeere Ingwer Kiwi Mango Nektarine
1 2 12 17 16 6 11 4 7
Banane Erdbeere Himbeere Ingwer Kiwi Mango Nektarine Pflaume Quitte
16 2 13 17 11 8 4 14 3
```

```
Banane Erdbeere Feige Grapefruit Himbeere Kiwi Litschi Orange Quitte
7 16 17 1 12 2 14 10 13 9
Apfel Banane Dattel Erdbeere Feige Kiwi Mango Nektarine Orange Pflaume
12 5 16 3 17 8 1 7 2 11 14
Banane Erdbeere Grapefruit Himbeere Johannisbeere Kiwi Litschi Mango Nektarine
    Orange Pflaume
12 8 16 14 4 15 1 5 13 10 6
Clementine Dattel Erdbeere Feige Grapefruit Ingwer Johannisbeere Mango Orange
    Pflaume Quitte

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Apfel: [9]
Feige: [13]
Grapefruit: [8]
Ingwer: [6]
Kiwi: [2]
Nektarine: [7]
Orange: [14]
Pflaume: [12]
Clementine: [15]
Erdbeere: [16]
Himbeere: [11]
Mango: [1]
Banane: [17]
Quitte: [4]
Litschi: [3]
Dattel: [10]
Johannisbeere: [5]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[2, 6, 7, 8, 9, 12, 13, 14]

--- Befehl ---
$ java A2Main # Beispiel 5

--- INPUT  ---
20
Apfel Banane Clementine Dattel Grapefruit Himbeere Mango Nektarine Orange
    Pflaume Quitte Sauerkirsche Tamarinde
4
16 15 14 13 6 9 7 17 2 3
Banane Dattel Johannisbeere Kiwi Litschi Nektarine Orange Quitte Rosine
    Sauerkirsche
2 1 6 19 9 3 5 10 7 12 4 16 15
Apfel Banane Dattel Grapefruit Himbeere Kiwi Litschi Mango Orange Pflaume Quitte
    Sauerkirsche Tamarinde
5 11 13 16 10 20 7 2 15
Clementine Himbeere Ingwer Johannisbeere Kiwi Litschi Orange Pflaume Sauerkirsche
```

```
3 7 8 13 9 12 5 11 15 17
Banane Erdbeere Himbeere Ingwer Johannisbeere Kiwi Litschi Quitte Rosine
    Tamarinde

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Apfel: [1, 4, 19]
Banane: [3, 9]
Clementine: [20]
Dattel: [6]
Grapefruit: [1, 4, 19]
Himbeere: [5]
Mango: [1, 4, 19]
Nektarine: [14]
Orange: [2, 16]
Pflaume: [10]
Quitte: [3, 9]
Sauerkirsche: [2, 16]
Tamarinde: [12]
Johannisbeere: [13]
Kiwi: [7, 15]
Litschi: [7, 15]
Rosine: [17]
Ingwer: [11]
Erdbeere: [8]
null: [18]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 2, 3, 4, 5, 6, 9, 10, 12, 14, 16, 19, 20]

--- Befehl ---
$ java A2Main # Beispiel 6

--- INPUT  ---
23
Clementine Erdbeere Himbeere Orange Quitte Rosine Ugli Vogelbeere
8
13 18 3 22 14 2 12 15 23 21 20 11 8
Apfel Banane Feige Himbeere Ingwer Johannisbeere Kiwi Litschi Nektarine Orange
    Rosine Ugli Weintraube
11 20 18 2 9 15 4 19 12 5
Dattel Himbeere Johannisbeere Nektarine Orange Pflaume Quitte Rosine Tamarinde
    Ugli
4 8 1 11 23 17 21 9 13 7 15 2 3 16
Apfel Banane Clementine Grapefruit Kiwi Litschi Mango Nektarine Pflaume Quitte
    Rosine Sauerkirsche Ugli Weintraube
1 5 9 17 19 8 3 7 22
Apfel Banane Clementine Dattel Feige Grapefruit Mango Pflaume Tamarinde
17 14 4 13 12 1 18 5 21 10 3
```

```
Apfel Dattel Erdbeere Grapefruit Himbeere Ingwer Johannisbeere Litschi Mango
    Quitte Weintraube
2 5 20 4 18 17 22 12 7 9 10 14 23 1
Clementine Dattel Erdbeere Feige Grapefruit Himbeere Ingwer Johannisbeere Kiwi
    Mango Nektarine Orange Pflaume Quitte
23 21 10 3 17 5 9 2 8 19 13 20 22
Apfel Banane Dattel Erdbeere Feige Grapefruit Kiwi Litschi Nektarine Orange
    Pflaume Tamarinde Weintraube
2 9 3 21 16 12 10 23 20 6 7 5
Apfel Clementine Dattel Erdbeere Johannisbeere Kiwi Nektarine Orange Pflaume
    Sauerkirsche Vogelbeere Weintraube

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Clementine: [7]
Erdbeere: [10]
Himbeere: [18]
Orange: [20]
Quitte: [4]
Rosine: [11, 15]
Ugli: [11, 15]
Vogelbeere: [6]
Apfel: [3]
Banane: [8]
Feige: [22]
Ingwer: [14]
Johannisbeere: [12]
Kiwi: [23]
Litschi: [13]
Nektarine: [2]
Weintraube: [21]
Dattel: [5]
Pflaume: [9]
Tamarinde: [19]
Grapefruit: [17]
Mango: [1]
Sauerkirsche: [16]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[4, 6, 7, 10, 11, 15, 18, 20]

--- Befehl ---
$ java A2Main # Beispiel 7

--- INPUT  ---
26
Apfel Clementine Dattel Grapefruit Mango Sauerkirsche Tamarinde Ugli Vogelbeere
    Xenia Yuzu Zitrone
4
```

```
3 22 15 1 20 4 14 7 2 11 26 8 10 13
Apfel Erdbeere Feige Grapefruit Himbeere Ingwer Kiwi Litschi Nektarine Orange
    Quitte Sauerkirsche Xenia Yuzu
5 20 24 25 26 7 18 3 23 10 15
Apfel Banane Clementine Erdbeere Grapefruit Litschi Nektarine Tamarinde Ugli
    Xenia Zitrone
24 10 14 9 1 8 21 20 15 3 26
Apfel Clementine Erdbeere Grapefruit Kiwi Litschi Pflaume Sauerkirsche
    Weintraube Xenia Yuzu
11 1 25 13 15 22 2 19 12 24 18
Banane Clementine Erdbeere Feige Himbeere Johannisbeere Kiwi Orange Quitte
    Rosine Ugli

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
Apfel: [3, 10, 20, 26]
Clementine: [24]
Dattel: [6, 16, 17]
Grapefruit: [3, 10, 20, 26]
Mango: [6, 16, 17]
Sauerkirsche: [8, 14]
Tamarinde: [5, 23]
Ugli: [18, 25]
Vogelbeere: [6, 16, 17]
Xenia: [3, 10, 20, 26]
Yuzu: [8, 14]
Zitrone: [5, 23]
Erdbeere: [15]
Feige: [2, 11, 13, 22]
Himbeere: [2, 11, 13, 22]
Ingwer: [4]
Kiwi: [1]
Litschi: [3, 10, 20, 26]
Nektarine: [7]
Orange: [2, 11, 13, 22]
Quitte: [2, 11, 13, 22]
Banane: [18, 25]
Pflaume: [9, 21]
Weintraube: [9, 21]
Johannisbeere: [12, 19]
Rosine: [12, 19]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[3, 5, 6, 8, 10, 14, 16, 17, 18, 20, 23, 24, 25, 26]
Achtung! Schüsseln könnten auch andere Obstsorten beinhalten! Kritisch sind
    folgende Gruppen:
[Apfel, Grapefruit, Xenia, Litschi] in [3, 10, 20, 26]
[Ugli, Banane] in [18, 25]
Diese Schüsseln und Obstsorten sollten genauer beobachtet werden!
```

Und noch eigene Beispiele, die zeigen, dass es nicht notwendig ist, jede Zuordnung genau zu kennen:

```
--- Befehl ---
$ java A2Main # Beispiel 1

--- INPUT  ---
10
A B C D E
1
1 3 5 7 9
A B C D E

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
A: [1, 3, 5, 7, 9]
B: [1, 3, 5, 7, 9]
C: [1, 3, 5, 7, 9]
D: [1, 3, 5, 7, 9]
E: [1, 3, 5, 7, 9]
null: [2, 4, 6, 8, 10]
null: [2, 4, 6, 8, 10]
null: [2, 4, 6, 8, 10]
null: [2, 4, 6, 8, 10]
null: [2, 4, 6, 8, 10]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 3, 5, 7, 9]

--- Befehl ---
$ java A2Main # Beispiel 2

--- INPUT  ---
10
A B C D E
1
4 5 6 8 9
F G H I J

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
A: [1, 2, 3, 7, 10]
B: [1, 2, 3, 7, 10]
C: [1, 2, 3, 7, 10]
D: [1, 2, 3, 7, 10]
E: [1, 2, 3, 7, 10]
F: [4, 5, 6, 8, 9]
G: [4, 5, 6, 8, 9]
H: [4, 5, 6, 8, 9]
```

```
I: [4, 5, 6, 8, 9]
J: [4, 5, 6, 8, 9]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 2, 3, 7, 10]
```

Und was ebenso passieren kann, wenn nicht alle Zuordnungen genau bekannt sind:

```
--- Befehl ---
$ java A2Main # Beispiel 3

--- INPUT  ---
10
A B C D E
1
2 4 5 6 8 9
E F G H I J

--- OUTPUT ---
Finaler Kenntnisstand bezüglich der Zuordnung:
A: [1, 3, 7, 10]
B: [1, 3, 7, 10]
C: [1, 3, 7, 10]
D: [1, 3, 7, 10]
E: [2, 4, 5, 6, 8, 9]
F: [2, 4, 5, 6, 8, 9]
G: [2, 4, 5, 6, 8, 9]
H: [2, 4, 5, 6, 8, 9]
I: [2, 4, 5, 6, 8, 9]
J: [2, 4, 5, 6, 8, 9]
Menge der Schüsseln, in denen sich die Wunschsorten befinden:
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Achtung! Schüsseln könnten auch andere Obstsorten beinhalten! Kritisch sind
    folgende Gruppen:
[E, F, G, H, I, J] in [2, 4, 5, 6, 8, 9]
Diese Schüsseln und Obstsorten sollten genauer beobachtet werden!
```

7 Quellcode

```
// Obstsortenregister (String > int, int > String)
Map<String, Integer> fruits = new HashMap<String, Integer>();
String[]             names;

// Obstsorte registrieren oder abrufen
private static int getFruit(String name) {
  Integer v = fruits.get(name);
  if (v == null) {
    int s = fruits.size();
    fruits.put(name, Integer.valueOf(s));
```

```
    names[s] = name;
    return s;
  }
  return v.intValue();
}

public static void main(String[] args) {
  // Anzahl der Obstsorten
  int size;
  // gewünschte Obstsorten
  int[] requests;
  // Anzahl der Beobachtungen
  int n;
  // Kenntnisstand
  Matrix fruitMatrix;

  [Einleseprozedur]

  // initialer Kenntnisstand: jedes Obst könnte sich in jeder Schüssel
  // befinden
  fruitMatrix = new Matrix(size);

  // Beobachtungen einlesen
  for (int i = 0; i < n; ++i) {
    // Nummern der Schüsseln
    int[] b;
    [Einleseprozedur]
    // Obstsorten
    int[] a;
    [Einleseprozedur]
    // Beobachtungsaussagen kombinieren
    Matrix observation = new Matrix(size, a, b);
    fruitMatrix.combine(observation);
  }
  // Menge der Wunschschüsseln
  int x = 0;
  // Bitmaske aller gewünschten Obstsorten
  int requestBitmask = 0;
  for (int i : requests) {
    requestBitmask |= 1 << i;
  }
  // weitere nötige Beobachtungen der Form int[] {<Obstgruppe als Bitmaske>,
  // <Schüsselgruppe als Bitmaske>}
  List<int[]> critical = new ArrayList<int[]>();
  int criticalMask = 0;
  for (int i : requests) {
    int bowls = fruitMatrix.getBowls(i);
    x |= bowls;
    int fruits = fruitMatrix.getFruits(bowls);
    // prüfen, ob in möglichen Schüsseln auch unerwünschte Obstsorten
    // enthalten sein können, d. h. ob in fruits Bits gesetzt sind, die nicht
    // in requestBitmask gesetzt sind -> aber nicht doppelt warnen
    if ((fruits | requestBitmask) != requestBitmask
        && (criticalMask & fruits) == 0) {
      critical.add(new int[] { fruits, bowls });
      criticalMask |= fruits;
    }
  }
```

```
  // Ausgabe der Ergebnisse
  System.out.println(
      "Menge der Schüsseln, in denen sich die Wunschsorten befinden:");
  System.out.println(Arrays.toString(bitMaskToArray(x)));
  if (Integer.bitCount(x) > requests.length) {
    System.out.println("Achtung! Schüsseln könnten auch andere Obstsorten "
        + "beinhalten! Kritisch sind folgende Gruppen:");
    for (int[] g : critical) {
      int[] fruitsInt = bitMaskToArray(g[0]);
      String[] fruitsString = new String[fruitsInt.length];
      for (int i = 0; i < fruitsString.length; ++i)
        fruitsString[i] = names[fruitsInt[i] - 1];
      System.out.println(Arrays.toString(fruitsString) + " in "
          + Arrays.toString(bitMaskToArray(g[1])));
    }
    System.out.println("Diese Schüsseln und Obstsorten "
        + "sollten genauer beobachtet werden!");
  }
}

// alle in mask gesetzten Bits als int[] zurückgeben
private static int[] bitMaskToArray(int mask) {
  int[] rs = new int[Integer.bitCount(mask)];
  for (int i = 0; mask > 0; ++i) {
    // kleinstes Bit ermitteln
    rs[i] = Integer.numberOfTrailingZeros(mask) + 1;
    // kleinstes Bit entfernen
    mask = mask ^ (mask & -mask);
  }
  return rs;
}

private static class Matrix {
  // zeilenweise als Bitmaske speichern
  private int[] data;

  // alles auf 1 setzen
  public Matrix(int size) {
    data = new int[size];
    int mask = (1 << size) - 1;
    for (int i = 0; i < size; ++i) data[i] = mask;
  }

  // Matrix für eine Beobachtung erstellen
  public Matrix(int size, int[] a, int[] b) {
    data = new int[size];
    // alle Schüsseln
    int all = (1 << size) - 1;
    int mask = 0;
    // alle Schüsseln in b markieren
    for (int x : b) mask |= 1 << x;
    // Die Schüsselmaske für alle Obstsorten in a ist mask (die Schüsseln in
    // b), für alle anderen ist es ~mask bezüglich all, also all ^ mask
    int inv = all ^ mask;
    // alle auf inv setzen
    for (int i = 0; i < size; ++i) data[i] = inv;
    // und die Obstsorten in a "zurück"setzen
    for (int x : a) data[x] = mask;
```

```
  }

  // Nummern der Schüsseln für ein bestimmtes Obst
  public int getBowls(int f) {
    return data[f];
  }

  // Obstsorten für eine Schüsselmenge
  public int getFruits(int bowlMask) {
    int rs = 0;
    for (int i = 0; i < data.length; ++i) {
      if ((data[i] & bowlMask) != 0) rs |= 1 << i;
    }
    return rs;
  }

  // komponentenweise multiplizieren (oder logisches &), um Aussagen zu
  // verknüpfen
  public void combine(Matrix o) {
    for (int i = 0; i < data.length; ++i) data[i] &= o.data[i];
  }
}
```