# Aufgabe 2: Spießgesellen

Teilnahme-Id: 57063

Bearbeiter/-in dieser Aufgabe:
Raphael Gaedtke

5. April 2021

## Inhaltsverzeichnis



# 1 Lösungsidee

Es wird zunächst ein Algorithmus angegeben, der eine Lösung für Teilaufgabe b) ist. Später wird dieser dann händisch auf die angegebene Beispielsituation in Teilaufgabe a) angewandt,

## 1.1 Allgemeines

**Definition 1** *Es sei $n$ die Anzahl der Schüsseln und $w$ die Anzahl der Obstsorten auf Donalds Wunschliste. Außerdem sei $O$ die Menge der verfügbaren Obstsorten, für die $|O| = n$ gilt.*

Im Folgenden sollen die bereitgestellten Schüsseln 1-indiziert, also von 1 bis $n$, durchnummeriert werden.

**Definition 2** *Es sei $a \in \mathbb{N}_{\leq n}$ die Nummer einer der aufgestellten Schüsseln. Dann sei $z_a$ die Menge der von Donald beobachteten Obstspieße, die unter Anderem aus der Schüssel mit der Nummer $a$ stammen. Die Menge $S_a \subset (O \times \mathbb{N} \cup \{0\})$ sei dann eine Menge von $n$ Tupeln, wobei jedes dieser Tupel eine Obstsorte enthält, sodass diese Sorten zwischen allen Tupeln paarweise verschieden sind. Außerdem enthält jedes dieser Tupel eine nichtnegative Ganzzahl, die angibt, wie oft sich die entsprechende Obstsorte auf einem unter anderem aus der Schüssel mit der Nummer $a$ stammenden Obstspieß befindet.*

**Beispiel 1** *In der auf dem Aufgabenblatt gegebenen Beispielsituation ist für die erste gegebene Schüssel $S_1 = \{(Apfel, 2), (Banane, 1), (Brombeere, 2), (Erdbeere, 1), (Pflaume, 0), (Weintraube, 0)\}$.*

**Definition 3** *Es sei $a \in \mathbb{N}_{\leq n}$ die Nummer einer der aufgestellten Schüsseln. Dann sei $M_a$ mit $M_a \subset O$ die Menge $M_a = \{o \in O | (o, z_a) \in S_a\}$.*

Für eine Zahl $a \in \mathbb{N}_{\leq n}$ ist $M_a$ die Menge der Obstsorten, die sich auf allen aus der Schüssel mit der Nummer $a$ stammenden Obstspießen befindet. In der Schüssel mit der Nummer $a$ können sich keine Obstsorten befinden, die nicht auch Element der Menge $M_a$ sind, weil sich die Sorte, die sich tatsächlich in der Schüssel mit der Nummer $a$ befindet, auf jedem Obstspieß aus dieser Schüssel befinden muss.

## 1.2 Der Algorithmus

Nach der Eingabe soll die Suche nach den Schüsseln mit Donalds Wunschsorten in drei Schritten ablaufen:

1. Bestimmen von $M_a$ für $\forall a \in \mathbb{N}_{\leq n}$.
2. Schließen auf den Inhalt der einzelnen Schüsseln
3. (Wenn möglich) Bestimmung der Schüsseln mit Donalds Wunschsorten

### 1.2.1 Bestimmen von $M_a$

Um $M_a$ für $\forall a \in \mathbb{N}_{\leq n}$ zu bestimmen, wird zunächst für jede Schüssel $a$ die Menge $S_a$ bestimmt. Dazu wird für jede Schüssel eine Liste für alle Obstsorten geführt, die zählt, wie oft sich jede Obstsorte auf einem Obstspieß aus dieser Schüssel befunden hat. Die Menge $S_a$ ist nach der Definition von $S_a$ dann die Menge aller Tupel, die aus einer Obstsorte und dem entsprechenden Listeneintrag der jeweiligen Schüssel gebildet werden.
Um $M_a$ zu erhalten, werden dann in $S_a$ alle Obstsorten ausgewählt, die nach der Zahl in „ihrem“ Tupel $z_a$-mal gezogen wurden, wobei auch $z_a = 0$ sein kann.
Weil der Fall $z_a = 0$ ebenfalls berücksichtigt wird, werden auch Schüsseln, aus denen nichts entnommen wurde, und Obstsorten, die vorhanden waren, aber nicht gezogen wurden, behandelt.

### 1.2.2 Schließen auf die Schüsselinhalte

Es gibt drei aufeinanderfolgende Schritte, mit denen auf die Inhalte der Schüsseln geschlossen wird. Es sei zunächst $A$ eine Obstsorte und $i$ und $j$ zwei positive Ganzzahlen mit $A \in M_i$ und $A \in M_j$. Wenn dann $z_i < z_j$ gilt, die Obstsorte $A$ also häufiger auf einem Obstspieß aus $j$ als aus $i$ war, kann in der Schüssel mit dem Index $i$ nicht die Obstsorte $A$ enthalten sein, weil diese Obstsorte sich auch auf mindestens einem Obstspieß befindet, der ohne diese Schüssel zusammengestellt wurde. Zu Beginn des Schlussverfahrens werden also alle Obstsorten, die sich in mehreren Schüsseln befinden könnten, aus allen Listen für Schüsseln entfernt, aus denen sie nicht mit maximaler Häufigkeit gezogen wurden.
Danach lässt sich eine Schüssel $a$ aus $M_a$ genau dann einer Obstsorte eindeutig zuordnen, wenn $|M_a| = 1$ ist. Das einzige Element von $M_a$ ist dann die in der Schüssel enthaltenen Obstsorte.
Basierend auf dieser Obstsorte können von der Menge aller Mengen $M_a$ aus weiterführende Schlüsse auf den Inhalt der einzelnen Schüsseln gezogen werden.
Wenn die Menge $M_a$ für ein $a$ nur ein Element hat und somit die Position einer Obstsorte bestimmt ist, ist der Inhalt der Schüssel $a$ gesichert und diese Schüssel kann aus der Betrachtung gestrichen werden. Gleichzeitig kann dann aber auch die Obstsorte, die sich sicher in der Schüssel $a$ befinden muss, aus allen anderen Mengen $M_b$ gestrichen werden, wodurch sich $|M_b|$ für diese Schüsseln um 1 verringert. In der Menge $M_a$ soll diese Obstsorte allerdings für den folgenden Schritt enthalten bleiben.
Dieser Schritt wird wiederholt, bis entweder alle Schüsseln aus der Betrachtung gestrichen sind, oder jede der noch nicht aus der Betrachtungen gestrichenen Schüsseln mehr als ein Element enthält.
Danach wird in einem letzten Schritt überprüft, ob es eine Obstsorte gibt, die nur für eine Nummer $a$ Element von $M_a$ ist, für die aber $M_a > a$ gilt. Diese Obstsorte lässt sich dann eindeutig dieser Schüssel zuordnen. Wie auch im Schritt davor, wird hier nach dem Auffinden einer Obstsorte immer wieder nach einer neuen Obstsorte mit dieser Eigenschaft gesucht, bis dies nicht mehr möglich ist.
Danach kann nicht weiter auf die möglichen Schüsselinhalte geschlossen werden.

### 1.2.3 Schüsseln mit Donalds Wunschsorten

Nachdem das in Kapitel 1.2.2 beschriebene Verfahren durchgeführt wurde, beschreibt die Menge $M_a$ für eine Schüssel $a$ die kleinstmögliche Menge von Obstsorten, die sich in dieser Schüssel befinden könnte, weil diese durch weitere Schlüsse auf Basis von Donalds bisher gemachten Beobachtungen nicht mehr

verkleinert werden kann.
Es sei $G$ dann die Menge aller Indizes $a$, für die $M_a$ eine Obstsorte auf Donalds Wunschliste enthält. Dann sei

$$W := \bigcup_{a \in G} M_a.$$

Wenn $|W| = w$ gilt, ist $G$ die Menge von Listen, an denen Donald sich anstellen muss, weil $W$ dann genau die Menge der Obstsorten auf Donalds Wunschliste ist. Somit wird die Menge $G$ als Ergebnis der Berechnung ausgegeben.
Andernfalls kann die Menge der Schüsseln, in denen sich Donalds Wunschsorten befinden, nicht eindeutig bestimmt werden. In diesem Fall gibt das Programm als möglichst informative Meldung die Menge $M_a$ für $\forall a \in \mathbb{N}_{\leq n}$ aus.
Diese entspricht genau der Menge $G$. Wenn Donald sich aus diesen Schüsseln bedient, erhält er mehr Obstsorten, als auf seiner Wunschliste stehen.

## 1.3 Laufzeitüberlegungen

Die Laufzeitüberlegungen für den in Kapitel 1.2 beschriebenen Algorithmus werden mit Landau-Symbolen ausgedrückt. Dazu wird der Algorithmus wiederum in die in Kapitel 1.2 gegebenen Schritte unterteilt, wobei wie bei den Beispielen auf der BwInf-Webseite $n \leq 26$ angenommen wird:

- Die Bestimmung der Mengen $S_a$ kann schon während des Einlesenes der Eingabedaten vollzogen werden, indem für jede Obstsorte auf einem Obstspieß eine der Menge von Schüsseln entsprechende Menge von Zählern um jeweils 1 erhöht wird. Da Ein- und Ausgabe bei den Laufzeitüberlegungen hier nicht berücksichtigt werden sollen und die Bestimmung der Mengen $S_a$ für $n \leq 26$ nur wenige Operationen in Anspruch nimmt, werden diese hier ebenfalls nicht berücksichtigt.

- Die Mengen $M_a$ können aus den Mengen $S_a$ erzeugt werden, indem jeder Eintrag jeder Menge $S_a$ mit $z_a$ abgeglichen wird. Für jede der $n$ Mengen $S_a$ gilt $|S_a| = n$, also ist die Laufzeit für diesen Schritt etwa $O(n^2)$.

- Das in Kapitel 1.2.2 beschriebene Schlussverfahren ist in drei Schritte aufgeteilt. Für den ersten Schritt muss jeder Eintrag einer jeden Menge $M_a$ durchgegangen werden, dieser Schritt also eine Laufzeit von $O(n^2)$.
  Im zweiten Schritt muss im schlechtesten Fall die Menge aller Mengen $M_a$ $n$-mal durchlaufen werden, woraus eine Laufzeit von $O(n^3)$ folgt, die auch der dritte Schritt hat. Für diese beiden Schritte ist die angegebene Laufzeit allerdings recht ungenau, da die Menge aller Mengen $M_a$ nicht immer komplett durchlaufen werden muss. Tatsächlich ist die Laufzeit des Schlussverfahrens, für das sich hier die Laufzeit $O(n^2 + 2n^3)$ ergibt, also geringer.

- Um $G$ zu erhalten, müssen wieder alle Mengen $M_a$ durchgegangen werden, was eine Laufzeit von $O(n^2)$ ergibt. Um die Menge $W$ zu bilden, wird dann wiederum eine Laufzeit von $O(n^2)$ benötigt, um alle Elemente in eine Menge zu schieben und eine Laufzeit von $O(\frac{n(n+1)}{2})$, um alle Duplikate aus dieser Liste zu entfernen. Insgesamt ergibt sich für diesen Schritt also eine Laufzeit von $O(2n^2 + \frac{n(n+1)}{2})$. Diese Angabe ist allerdings wiederum eine recht ungenaue Obergrenze.

Insgesamt ergibt sich eine maximale Laufzeit von $O(4n^2 + \frac{n(n+1)}{2}) + n^3)$. Dies ist eine polynomielle Laufzeit, wobei der Algorithmus mit $n \leq 26$ schnell genug arbeitet.

## 1.4 Lösung für Teilaufgabe a)

Der in Kapitel 1.2 beschriebene Algorithmus wird nun auf die gegebene Beispielsituation angewandt. In dieser Situation ist $n = 6$, $w = 3$ und $O = \{Apfel, Banane, Brombeere, Erdbeere, Pflaume, Weintraube\}$. Dafür müssen zunächst die Mengen $S_1$, $S_2$, $S_3$, $S_4$, $S_5$ und $S_6$ bestimmt werden:

$$S_1 = \{(Apfel, 2), (Banane, 1), (Brombeere, 2), (Erdbeere, 1), (Pflaume, 0), (Weintraube, 0)\}$$

$$S_2 = \{(Apfel, 1), (Banane, 0), (Brombeere, 1), (Erdbeere, 2), (Pflaume, 1), (Weintraube, 0)\}$$

$$S_3 = \{(Apfel, 0), (Banane, 1), (Brombeere, 0), (Erdbeere, 0), (Pflaume, 1), (Weintraube, 1)\}$$

$$S_4 = \{(Apfel, 2), (Banane, 1), (Brombeere, 2), (Erdbeere, 1), (Pflaume, 0), (Weintraube, 0)\}$$

$$S_5 = \{(Apfel, 1), (Banane, 2), (Brombeere, 1), (Erdbeere, 0), (Pflaume, 1), (Weintraube, 1)\}$$
$$S_6 = \{(Apfel, 0), (Banane, 1), (Brombeere, 1), (Erdbeere, 1), (Pflaume, 2), (Weintraube, 1)\}$$

Außerdem ergibt sich für $z_1$, $z_2$, $z_3$, $z_4$, $z_5$ und $z_6$:

| $a$ | $z_a$ |
|---|---|
| 1 | 2 |
| 2 | 2 |
| 3 | 1 |
| 4 | 2 |
| 5 | 2 |
| 6 | 2 |

Darauf aufbauend können die Mengen $M_1$, $M_2$, $M_3$, $M_4$, $M_5$ und $M_6$ bestimmt werden:

$$M_1 = \{Apfel, Brombeere\}$$
$$M_2 = \{Erdbeere\}$$
$$M_3 = \{Banane, Pflaume, Weintraube\}$$
$$M_4 = \{Apfel, Brombeere\}$$
$$M_5 = \{Banane\}$$
$$M_6 = \{Pflaume\}$$

Auf diese Mengen wird das in Kapitel 1.2.2 beschriebene Schlussverfahren angewendet, wobei der erste und der dritte Schritt dieses Verfahrens überhaupt nicht angewendet werden müssen bzw. können.

$$M_1 = \{Apfel, Brombeere\}$$
$$M_2 = \{Erdbeere\}$$
$$M_3 = \{Weintraube\}$$
$$M_4 = \{Apfel, Brombeere\}$$
$$M_5 = \{Banane\}$$
$$M_6 = \{Pflaume\}$$

Da Donalds Wunschliste die Obstsorten Weintraube, Brombeere und Apfel enthält, ist die Menge $G$ der Indizes von Mengen $M_a$, die mindestens eine dieser Obstsorten enthalten, eben $G := \{1, 3, 4\}$.
Für die Menge $W$ gilt dann

$$W = \bigcup_{a \in G} M_a = M_1 \cup M_3 \cup M_4 = \{Apfel, Brombeere, Weintraube\}.$$

Weil in diesem Fall $|W| = 3 = w$ gilt, ist $G$ hier die Menge der Schüsseln, an denen Donald sich bedienen muss, um einen Obstspieß mit den Obstsorten Apfel. Brombeere und Weintraube zu erhalten.
Donald sollte sich also an den Schlangen vor den Schüsseln mit den Nummern 1, 3 und 4 anstellen.

# 2 Umsetzung

Die Lösungsidee wird in C++ umgesetzt.

## 2.1 Einlesen und Speicherung der Eingabedaten

Jeder Obstsorte soll ein Index zwischen 0 und $n-1$ zugeordnet werden. Um dies bei der Eingabe umzusetzen, wird ein Container des Typs map<string, int> mit der Bezeichnung„indizesObstsorten“ gespeichert. Bei der Eingabe liefert die Funktion „indexObstsorte()“ dann den Index einer Obstsorte zurück, indem sie ihn aus dem Container „indizesObstsorten“ ausliest oder ihr einen neuen Index zuordnet, falls die betreffende Obstsorte zum ersten Mal abgefragt wird.
Um später die Ausgabe zu erleichtern, wird in einem Container des Typs map<int, string> mit der Bezeichnung „namenObstsorten“ jedem Index einer Obstsorte wieder deren Name zugeordnet. Die beiden Container „namenObstsorten“ und „indizesObstsorten“ werden von der Funktion „indexObstsorten“ initialisiert und aktualisiert.
Alle Mengen $S_a$ für $a \in \mathbb{N}_{\leq n}$ werden gemeinsam im Container „SMengen“ gespeichert. Dieser ist ein Container des Typs vector<vector<int> > und speichert für jede der $n$ Mengen $S_a$ genau $n$ Zahlen in einem Container. Die Zahl mit dem Index $i$ gibt dann an, wie oft die Obstsorte mit dem Index $i$ aus der entsprechenden Schüssel gezogen wurde.
Zudem speichert der Container „ziehungen“, der vom Typ vector<int> ist, für jede Schüssel, wie oft ein Obstspieß unter anderem aus ihrem Inhalt zusammengestellt wurde, also für eine Schüssel mit dem Index $a$ die Zahl $z_a$. Zusätzlich werden in dem Container „wunschsorten“, der vom gleichen Typ ist, Donalds Wunschsorten eingelesen.
Alle dieser Variablen werden in der Funktion „datenEinlesen“ initialisiert. Im ganzen Programm sind die Indizes der Schüsseln, anders als in der von Daisy erzeugten Situation, 0-indiziert.

## 2.2 Konstruktion und Speicherung aller Mengen $M_a$

Alle Mengen $M_a$ für $a \in \mathbb{N}$ werden wiederum in einem Container des Typs vector<vector<int> > gespeichert. Dieser hat die Bezeichnung „MMengen“, wobei der Container des Typs vector<int> mit dem Index $i$ die Menge $M_i$ speichert. Die Schüsseln werden 0-indiziert durchnummeriert.
Der Container „MMengen“ wird in zwei ineinander verschachtelten for-Schleifen initialisiert, die den Container „SMengen“ durchlaufen und für jeden Eintrag in einer Menge $S_b$, der gleich der Zahl $z_b$ ist, den entsprechenden Index einer Obstsorte in den die Menge $M_b$ repräsentierenden Container einfügen.

## 2.3 Implementierung des Schlussverfahrens

Das in Kapitel 1.2.2 beschriebene Schlussverfahren wird auf dem Container „MMengen“ ausgeführt und bildet auch die drei dort beschriebenen Schritte ab.
Der erste dieser Schritte soll noch vor der Konstruktion aller Mengen $M_a$ ablaufen. Dabei wird der Container „SMengen“ in zwei ineinander verschachtelten for-Schleifen durchlaufen und die Anzahl der Ziehungen einer Obstsorte in einer Schüssel, in der diese Anzahl nicht maximal ist, auf -1 gesetzt. Dadurch wird sie bei der Konstruktion aller Mengen $M_a$ dann nicht übernommen.
Für jeden Index einer $i$ Menge $M_i$ wird im zweiten Schritt in einem Container vom Typ vector<bool> mit der Bezeichnung „inBetrachtung“ zunächst festgehalten, ob die jeweilige Menge aus der Betrachtung gestrichen ist.
Der Container wird dann in einer for-Schleife durchlaufen, um Indizes $i$ mit $|M_i| = 1$ zu finden. Dabei werden nur Container untersucht, die laut dem Speicher „inBetrachtung“ noch nicht bestimmt sind. Wird ein solcher Index gefunden, wird die Menge $M_i$ im Container „inBetrachtung“ als feststehend markiert und später nicht berücksichtigt.
Danach wird das erste und einzige Element der Menge $M_i$ aus allen anderen Containern entfernt, wozu der Container „MMengen“ in zwei verschachtelten for-Schleifen durchlaufen wird.
Danach wird der Zähler der for-Schleife zurückgesetzt, um nach dem nächsten Index $i$ mit $|M_i| = 1$ zu suchen.Sobald die for-Schleife einmal durch den ganzen Container durchgelaufen ist, kann dieser Schritt nicht erneut angewandt werden.
Für den dritten Schritt wird ebenfalls der Container „inBetrachtung“ der zuvor zurückgesetzt wird, verwendet, um festzustellen, ob eine Schüssel noch betrachtet werden muss. Dabei wird der Container „MMengen“ wiederum in zwei ineinander verschachtelten for-Schleifen durchlaufen. Für jede Obstsorte wird in einem Container vom Typ „vector<int>“ festgehalten, wie oft sie dabei vorkommt. In einem

weiteren Container des gleichen Typs wird zudem der Index der jeweils letzten Schüssel gespeichert, in der diese Obstsorte gefunden wurde.
Am Ende der for-Schleife wird überprüft, ob Obstsorten nur einmal gezählt wurden. In diesem Fall werden die entsprechenden Container mit nur diesem einen Element initialisiert und ihre Indizes aus der Betrachtung gestrichen. Andernfalls ist das Schlussverfahren beendet.

### 2.4 Konstruktion der Menge $W$ und Ausgabe

Wie in Kapitel 1.2.3 beschrieben, wird zunächst die Menge G konstruiert. Diese Menge soll dann die Indizes $i$ aller Mengen $M_i$ enthalten, die mindestens eine von Donalds Wunschsorten enthalten.
Um diese Menge zu bestimmen, wird der Container „MMengen“ in zwei ineinander verschachtelten for-Schleifen durchlaufen, wobei für jedes Element überprüft wird, ob es auch im Container „wunschsorten“ enthalten ist. Ist dies der Fall, wird der Index der Menge $M_i$ in den Container „MengeG“, der vom Typ vector<int> ist, geschoben.
Danach wird die Vereinigung aller Mengen $M_i$, deren Index im Container „MengeG“ enthalten sind, konstruiert und im Container „MengeW“, der vom Typ vector<int> ist, gespeichert. Dieser Container entspricht dann der Menge $W$ aus Kapitel 1.2. Um diese Menge zu konstruieren, wird zunächst der Container „MengeG“ in einer for-Schleife durchlaufen und jede Menge mit einem darin enthaltenen Index an den Container „MengeW“ angehängt. Danach wird dieser Container ein weiteres Mal in zwei verschachtelten for-Schleifen durchlaufen, um mehrfach enthaltene Elemente zu entfernen.
Ist dies geschehen, kann überprüft werden, ob $|W| = w$ gilt, und eine entsprechende Ausgabe gemacht. Ist dies der Fall, wird die im Container „MengeG“ enthaltene Menge von Schüsseln ausgegeben, andernfalls werden alle Mengen $M_i$ ausgegeben.

## 3 Beispiele

Im Folgenden werden die Programmausgabe für die auf der BwInf-Webseite gegebenen Eingabedaten aufgelistet. Die Eingabedatei spiesse0.txt enthält zusätzlich die auf dem Aufgabenblatt gegebene und in Teilaufgabe a) gelöste Situation:

Die Eingabedatei spiesse0.txt:

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

Ausgabe für die Eingabedatei spiesse0.txt:

```
Donald hat genug Obstspiesse beobachtet, um sicher wissen zu koennen, an welchen Schuesseln er sich
anstellen muss!
Schuesseln, in denen sich seine Wunschsorten befinden: 1 3 4
```

Ausgabe für die Eingabedatei spiesse1.txt:

```
Donald hat genug Obstspiesse beobachtet, um sicher wissen zu koennen, an welchen Schuesseln er sich
anstellen muss!
Schuesseln, in denen sich seine Wunschsorten befinden: 1 2 4 5 7
```

Ausgabe für die Eingabedatei spiesse2.txt:

Donald hat genug Obstspiesse beobachtet, um sicher wissen zu koennen, an welchen Schuesseln er sich anstellen muss!
Schuesseln, in denen sich seine Wunschsorten befinden: 1 5 6 7 10 11

Ausgabe für die Eingabedatei spiesse3.txt:

Donald hat leider nicht genug Obstspiesse beobachtet. Es kann nicht eindeutig festgestellt werden, an welchen Schuesseln er sich anstellen muss.
Im Folgenden wird fuer jede Schuessel angegeben, welche Obstsorten sich in ihr befinden koennten. Diese Mengen sind so klein wie moeglich.
Schuessel Nr. 1: Himbeere
Schuessel Nr. 2: Litschi Grapefruit
Schuessel Nr. 3: Orange
Schuessel Nr. 4: Nektarine
Schuessel Nr. 5: Clementine
Schuessel Nr. 6: Apfel Banane
Schuessel Nr. 7: Feige Ingwer
Schuessel Nr. 8: Erdbeere
Schuessel Nr. 9: Johannisbeere
Schuessel Nr. 10: Feige Ingwer
Schuessel Nr. 11: Litschi Grapefruit
Schuessel Nr. 12: Kiwi
Schuessel Nr. 13: Dattel
Schuessel Nr. 14: Apfel Banane
Schuessel Nr. 15: Es ist keine Obstsorte bekannt, die sich in dieser Schuessel befinden koennte.

Ausgabe für die Eingabedatei spiesse4.txt:

Donald hat genug Obstspiesse beobachtet, um sicher wissen zu koennen, an welchen Schuesseln er sich anstellen muss!
Schuesseln, in denen sich seine Wunschsorten befinden: 2 6 7 8 9 12 13 14

Ausgabe für die Eingabedatei spiesse5.txt:

Donald hat genug Obstspiesse beobachtet, um sicher wissen zu koennen, an welchen Schuesseln er sich anstellen muss!
Schuesseln, in denen sich seine Wunschsorten befinden: 1 2 3 4 5 6 9 10 12 14 16 19 20

Ausgabe für die Eingabedatei spiesse6.txt:

Donald hat genug Obstspiesse beobachtet, um sicher wissen zu koennen, an welchen Schuesseln er sich anstellen muss!
Schuesseln, in denen sich seine Wunschsorten befinden: 4 6 7 10 11 15 18 20

Ausgabe für die Eingabedatei spiesse7.txt:

Donald hat leider nicht genug Obstspiesse beobachtet. Es kann nicht eindeutig festgestellt werden, an welchen Schuesseln er sich anstellen muss.
Im Folgenden wird fuer jede Schuessel angegeben, welche Obstsorten sich in ihr befinden koennten. Diese Mengen sind so klein wie moeglich.
Schuessel Nr. 1: Kiwi
Schuessel Nr. 2: Feige Himbeere Orange Quitte
Schuessel Nr. 3: Apfel Grapefruit Xenia Litschi
Schuessel Nr. 4: Ingwer
Schuessel Nr. 5: Tamarinde Zitrone
Schuessel Nr. 6: Dattel Mango Vogelbeere
Schuessel Nr. 7: Nektarine
Schuessel Nr. 8: Sauerkirsche Yuzu

Schuessel Nr. 9: Pflaume Weintraube
Schuessel Nr. 10: Apfel Grapefruit Xenia Litschi
Schuessel Nr. 11: Feige Himbeere Orange Quitte
Schuessel Nr. 12: Johannisbeere Rosine
Schuessel Nr. 13: Feige Himbeere Orange Quitte
Schuessel Nr. 14: Sauerkirsche Yuzu
Schuessel Nr. 15: Erdbeere
Schuessel Nr. 16: Dattel Mango Vogelbeere
Schuessel Nr. 17: Dattel Mango Vogelbeere
Schuessel Nr. 18: Ugli Banane
Schuessel Nr. 19: Johannisbeere Rosine
Schuessel Nr. 20: Apfel Grapefruit Xenia Litschi
Schuessel Nr. 21: Pflaume Weintraube
Schuessel Nr. 22: Feige Himbeere Orange Quitte
Schuessel Nr. 23: Tamarinde Zitrone
Schuessel Nr. 24: Clementine
Schuessel Nr. 25: Ugli Banane
Schuessel Nr. 26: Apfel Grapefruit Xenia Litschi

# 4 Quellcode

Die Funktion „indexObstsorte“, die den Index einer Obstsorte zurückliefert und gegebenenfalls die Container „namenObtsorten“ und „indizesObstsorten“ initialisiert:

```
int indexObstsorte(string sorte, map<string, int>* indizesObstsorten,
map<int, string>* namenObstsorten)
{
    //Diese Funktion liefert den Index einer Obstsorte zurueck. Ist diese Sorte bisher noch
    //unbekannt, aktualisiert sie zusaetzlich die via Zeiger uebergebenen Container
    int groesse = (int) indizesObstsorten->size();
    //Groesse der Container vor dem Aufruf dieser Funktion
    int index = (*indizesObstsorten)[sorte];
    if((int) indizesObstsorten->size() == groesse)
    {
        //Der Obstsorte ist schon ein Index zugeordnet
        return index;
    }
    else
    {
        //Der Obstsorte muss noch ein Index zugeordnet werden!
        (*indizesObstsorten)[sorte] = groesse;
        (*namenObstsorten)[groesse] = sorte;
        //Die Container sind aktualisiert
    }
    return groesse;
}
```

Die main-Funktion des Programms, in der der Algorithmus implementiert wurde:

```
int main()
{
    //Variablen zum Speichern der Eingabedatei:
    map<string, int> indizesObstsorten; //Ordnet den Namen einer Obstsorte deren Index zu
    map<int, string> namenObstsorten; //Ordnet den Index einer Obstsorte deren Name zu
    int anzahlObstsorten; //Anzahl der Obstsorten
    vector<vector<int>> SMengen (0, vector<int> (0));
    //Container fuer Mengen des Typs S_a (s. Dokumentation)
    vector<int> ziehungen (0); //Speichert fuer jede Schuessel a die Zahl z_a (s. Dokumentation)
    vector<int> wunschsorten (0); //Donalds Wunschsorten
    string dateinameAusgabedatei; //Name der Ausgabedatei
    //Einlesen der Eingabedaten:
    datenEinlesen(&indizesObstsorten, &namenObstsorten, &anzahlObstsorten, &SMengen, &ziehungen,
     &wunschsorten, &dateinameAusgabedatei);

    //Der erste Schritt des Schlussverfahrens:
    for(int i = 0; i < anzahlObstsorten; i++) //Durch alle Obstsorten gehen
```

```
18      {
20          //Maximum der Ziehungen dieser Obstsorte bestimmen:
            int maximaleZiehungszahl = 0;
22          for(int j = 0; j < anzahlObstsorten; j++)
            {
24              if(SMengen[j][i] > maximaleZiehungszahl)
                {
26                  maximaleZiehungszahl = SMengen[j][i];
                }
            }
28
            //Nicht brauchbare Zahlen auf -1 setzten:_
30          for(int j = 0; j < anzahlObstsorten; j++)
            {
32              if(SMengen[j][i] < maximaleZiehungszahl)
                {
34                  SMengen[j][i] = -1;
                }
36          }
        }
38
        vector<vector<int>> MMengen (anzahlObstsorten, vector<int> (0));
40      //Container fuer Mengen des Typs M_a (s. Dokumentation)
        //Initialisieren der Containers MMengen:
42      for(int i = 0; i < anzahlObstsorten; i++)
        {
44          for(int j = 0; j < anzahlObstsorten; j++)
            {
46              if(SMengen[i][j] == ziehungen[i])
                {
48                  //Wird in diesen Anweisungsblock gesprungen, war die Obstsorte j auf jedem
                    // Obstspiess aus der Schuessel i!
50                  MMengen[i].push_back(j);
                }
52          }
        }
54
        //Der zweite Schritt des Schlussverfahrens:
56      //Container, der festhaelt, ob eine Menge noch betrachtet wird:
        vector<bool> inBetrachtung (0);
58      for(int i = 0; i < anzahlObstsorten; i++)
        {
60          inBetrachtung.push_back(true);
        }
62
        //Durchlaufen des Containers MMengen:
64      for(int i = 0; i < 26; i++)
        {
66          if(inBetrachtung[i] && MMengen[i].size() == 1)
            {
68              //Der Schuessel mit dem Index i kann eindeutig eine Obstsorte zugeordnet werden!
                inBetrachtung[i] = false; //Diese Schuessel aus der Betrachtung streichen
70              //Streichen der Obstsorte in dieser Schuessel aus allen anderen Mengen:
                for(int j = 0; j < (int) MMengen.size(); j++)
72              {
                    for(int k = 0; inBetrachtung[j] && k < (int) MMengen[j].size(); k++)
74                  {
                        if(MMengen[j][k] == MMengen[i][0])
76                      {
                            MMengen[j].erase(MMengen[j].begin() + k);
78                          break;
                        }
80                  }
                }
82              //Zaehler der for-Schleife zuruecksetzen:
                i = -1;
84          }
        }
86
        //Der dritte Schritt des Schlussverfahrens:
88
        vector<int> vorhandenObstsorten (anzahlObstsorten);
90      //Dieser Container zaehlt, wie oft jede Obstsorte gefunden wurde
```

```
    vector<int> schuesselIndizes (anzahlObstsorten);
92  //Dieser Container speichert fuer jede Obstsorte, in welcher Schuessel sie zuletzt gefunden wurde
    //Container inBetrachtung zuruecksetzen:
94  for(int i = 0; i < anzahlObstsorten; i++)
    {
96      inBetrachtung[i] = true;
        vorhandenObstsorten[i] = 0;
98  }

100 for(int i = 0; i < anzahlObstsorten; i++) //Durch MMengen gehen
    {
102     for(int j = 0; j < (int) MMengen[i].size() && inBetrachtung[i]; j++)
        {
104         vorhandenObstsorten[MMengen[i][j]]++;
            schuesselIndizes[MMengen[i][j]] = i;
106     }

108     if(i == anzahlObstsorten-1)
        {
110         //Der gesamte Container MMengen wurde durchlaufen!
            for(int j = 0; j < anzahlObstsorten; j++) //Durch Obstsorten gehen
112         {
                if(vorhandenObstsorten[j] == 1)
114             {
                    MMengen[schuesselIndizes[j]].clear();
116                 MMengen[schuesselIndizes[j]].push_back(j);
                    inBetrachtung[schuesselIndizes[j]] = false;
118             }
            }
120         //Container vorhandenObstsorten zuruecksetzen:
            for(int j = 0; j < anzahlObstsorten; j++)
122         {
                vorhandenObstsorten[j] = 0;
124         }
        }
126 }

128 //Nun wird die Menge G aller Schuesseln, die mindestens eine von Donalds Wunschsorten
    // enthalten koennten, bestimmt:
130 vector<int> MengeG (0); //Container fuer die Menge G
    for(int i = 0; i < anzahlObstsorten; i++) //Durchlaufen des Containers MMengen
132 {
        for(int j : MMengen[i])
134     {
            if(find(wunschsorten.begin(), wunschsorten.end(), j) != wunschsorten.end())
136         {
                //Eine der moeglichen Obstsorten fuer diese Schuesseln
138             //ist auch eine von Donalds Wunschsorten
                MengeG.push_back(i); //Index i der Menge G hinzufuegen
140             break;
            }
142     }
    }
144
    //Konstruktion der Menge W
146 vector<int> MengeW (0);
    for(int i : MengeG) //durchlaufen der Menge G
148 {
        for(int j : MMengen[i])
150     {
            MengeW.push_back(j);
152     }
    }
154
    //Entfernen doppelter Elemente aus der Menge W:
156 for(int i = 0; i < (int) MengeW.size(); i++)
    {
158     for(int j = i+1; j < (int) MengeW.size(); j++)
        {
160         if(MengeW[i] == MengeW[j])
            {
162             MengeW.erase(MengeW.begin()+j);
                j--;
```

```
164                 }
            }
166     }

168     if(MengeW.size() == wunschsorten.size())
        {
170         //Ausgabe (hier gekuerzt)
            return 0;
172     }
        else
174     {
           //Ausgabe (hier gekuerzt)
176     }

178     return 0;
}
```