Aufgabe 2: Spießgesellen

Teilnahme-Id: 55628

Bearbeiter dieser Aufgabe:
Michal Boron

April 2021

Inhaltsverzeichnis

1 Lösungsidee

1.1 Formulierung des Problems

Axiom 1. *Jeder **Obstsorte** wird genau ein einzigartiger natürlicher Index zugewiesen.*
Man schreibt: $o(x, i)$ — eine Obstsorte x besitzt einen Index i.

Gegeben sind eine Menge von n Obstsorten A und eine Menge von n ganzen Zahlen $B = \{1, 2, ..., n\}$, zu der die Indizes der Obstsorten aus A gehören.

Definition 1 (Spießkombination)**.** *Als eine **Spießkombination** $K = (F, Z)$ bezeichnet man eine Verknüpfung von zwei Mengen $F \subseteq A$ und Z, wobei $Z = \{i \in B \mid \forall x \in F : o(x, i)\}$.*

Gegeben sind auch m Spießkombinationen, wobei jede i–te Spießkombination aus einer Menge von Obstsorten $F_i \subseteq A$ und einer Menge der Zahlen $Z_i \subseteq B$ besteht. Nach der Definition 1 besteht die Menge Z_i nur aus den in B enthaltenen Indizes, die zu den Obstsorten in F_i gehören, deshalb sind die beiden Mengen F_i und Z_i auch gleichmächtig.

Außerdem gegeben ist auch eine ***Wunschliste*** $W \subseteq A$.

Die Aufgabe ist, zu entscheiden, ob die Menge der Indizes der in W enthaltenen Obstsorten $W' \subseteq B$ anhand der m Spießkombinationen eindeutig bestimmt werden kann. Falls ja, soll sie auch ausgegebn werden.

In den folgenden Überlegungen wird angenommen, dass das Axiom 1 für alle Obstsorten in der Eingabe gilt. Es ist aber möglich, dass die Spießkombination in einer Eingabe diesem Axiom nicht folgen, das heißt, es an einer Stelle einen Widersprcuh gibt. Laut der Aufgabenstellung ist ein solcher Fall nicht ausgeschlossen. Um diesen Fall zu verhindern, muss man die Korrektheit der Eingabe überprüfen. Mehr dazu folgt im Teil 1.5.

1.2 Bipartiter Graph

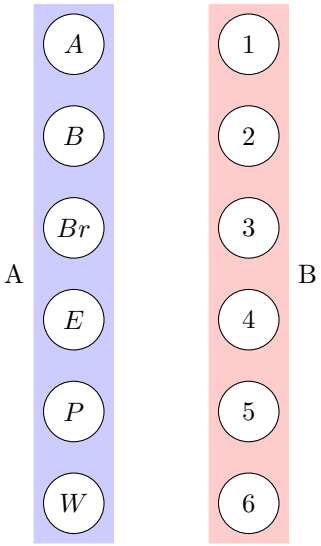
TODO: use definitions[?]

Man kann die beiden Mengen A und B zu Knoten eines bipartiten Graphen $G = (A \cup B = V, E)$ umwandeln. Die Menge der Kanten E wird im Folgenden festgelegt. Man stellt den Graphen als eine Adjazenzmatrix M der Größe $n \times n$ dar. Als M_i bezeichnet wird die Liste der Länge n, die die Beziehungen eines Knotens $i \in A$ zu jedem Knoten $j \in B$ als 1 (Kante) oder 0 (keine Kante) darstellt. Als $M_{i,j}$ bezeichnet wird die j–te Stelle in der i–ten Liste der Matrix.
Nach Axiom 1 gehört jeder Obstsorte aus A genau ein Index aus B. Dennoch man kann am Anfang keiner Obsorte einen Index zuweisen. Deshalb wird zunächst jeder Knoten aus A mit jedem Knoten aus B durch eine Kante verbunden:

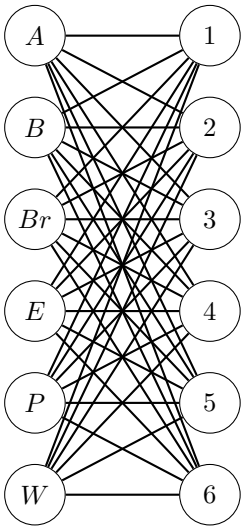
$$E = A \times B = \{(x, y) \mid x \in A \text{ und } y \in B\}.$$

Am Anfang ist M dementsprechend voll mit 1–en. Bei der Erstellung der Adjazenzmatrix kann man den Vorteil nutzen, dass die jeweilige Liste von Nachbarn des jeden Knotens $x \in A$ nur aus 0–en und 1-en besteht, indem diese Liste als Bitmasken dargestellt werden kann (mehr dazu in der Umsetzung).

Abbildung 1: Beide Abbildungen stellen den Graphen für das Beispiel aus der Aufgabenstellung dar. Die Buchstaben stehen für die entsprechenden Obstsorten aus diesem Beispiel (s. auch 3.1).



(a) Die entsprechenden Mengen des Graphen



(b) Der Graph am Anfang

Jede i–te Spießkombination bringt mit sich Informationen über die Obstsorten in F_i. Man kann Folgendes festellen.

Lemma 1. *Sei $K = (F, Z)$ eine Spießkombination. Für jede Obstsorte $o(x, i)$, wobei $x \in F$, gilt:*

(i) $i \in Z$.

(ii) $i \notin B \setminus Z$.

Deshalb darf man alle Kanten, die aus jedem Knoten $x \in F$ zu jedem Knoten $y \in B \setminus Z$ führen, aus E entfernen und nur die Kanten lassen, die zu allen $z \in Z$ führen.

Beweis. Nach Definition 1 gilt (i). Nach Axiom 1 besitzt jede Obstsorte einen einzigartigen Index i, deshalb kann i nicht gleichzeitig zu Z und $B \setminus Z$ gehören (ii). □

Aus Lemma 1 ergibt sich direkt aus eine andere Bemerkung.

Lemma 2. *Sei $C = (V_c, E_c) \subseteq G$ eine Zusammenhangskomponente. Sei $K = (F, Z)$ eine Spießkombiantion. Falls $F \cup Z \subsetneq V_c$ gilt, dann gilt auch:*

(i) $\forall p \in F, o(p, i) : i \in Z$,

(ii) $\forall q \in A \cap (V_c \cap F), o(q, j) : j \in B \cap (V_c \cap Z)$.

Deshalb werden alle Kanten, die aus jedem Knoten $x \in F$ zu jedem Knoten $y \in B \cap (V_c \setminus Z)$ führen, aus E entfernt und nur die Kanten lassen, die zu allen $z \in Z$ führen.

Beweis.
TODO: Beweis

□

Da Bitmasken für die Darstellung jeder Liste M_i ($i \in A$) verwendet werden, kann die Laufzeit bei der Analyse der jeweiligen Spießkombination optimiert werden (mehr dazu im Teil 1.6), weil man für die Operation des Entfernens Logikgatter verwenden kann.

1.3 Logik

Betrachten wir eine Spießkombination $s = (F_s, Z_s)$. Wir erstellen 3 Bitmasken bf, bn und br jeweils der Länge n. Die Bitmaske bf besteht aus n 1–en. In der Maske bn stehen die 1–Bits an allen Stellen, die den Indizes in Z_s entsprechen. Die Bitmaske br wird auf folgende Weise definiert:

$$br := \neg(bn) \wedge bf.$$

So können wir auf allen Listen M_i, wobei $i \in F_s$, die AND–Operation mit der Maske bn durchführen:

$$M_i := M_i \wedge bn.$$

Analog führen wir die AND–Operation mit der Maske br auf allen Listen M_j, wobei $j \in A \setminus F_s$, durch:

$$M_j := M_j \wedge br.$$

Abbildung 3: Beide Abbildungen stellen die Adjazenzmatrix für das Beispiel aus der Aufgabenstellung dar. Die Buchstaben in der ersten Spalte stehen für die entsprechenden Obstsorten und die Zahlen in der ersten Zeile stehen für die Indizes aus demselben Beispiel (s. auch 3.1). Auf der Abb. 4b stehen bn und br für die entsprechenden Bitmasken.

Spießkombination: F = {Banane, Pflaume, Weintraube}
Z = {3, 5, 6}

	6	5	4	3	2	1
bn	1	1	0	1	0	0
br	0	0	1	0	1	1

	6	5	4	3	2	1
A	0	1	1	0	0	1
B	0	1	1	0	0	1
Br	0	1	1	0	0	1
E	1	0	0	1	1	0
P	1	0	0	1	1	0
W	1	0	0	1	1	0

(a) M vor der neuen Spießkombination

	6	5	4	3	2	1
A	0	0	1	0	0	1
B	0	1	0	0	0	0
Br	0	0	1	0	0	1
E	0	0	0	0	1	0
P	1	0	0	1	0	0
W	1	0	0	1	0	0

(b) M nach der Verarbeitung der beschriebenen Spießkombination.

Auf der obigen Abbildung werden blau und rot die entsprechenden Listen gekennzeichnet, auf denen die AND–Operation mit der entsprechenden Bitmaske durchgefüht wurde. Rot werden die Bits gekennzeichnet, die sich nach der Verarbeitung der Spießkombination veränderten.

Was die beschriebenen Operationen verursachen, wird anhand der folgenden Fallunterscheidung erläutert.

1. Falls es sich um einen Knoten $x \in F_s$ handelt, betrachten wir dazu die entsprechende Liste M_x und einen Knoten $y \in B$.
   a) Falls der Knoten y zu Z_s gehört, aber an der Stelle $M_{x,y}$ 0 steht, bleibt es auch 0.
   b) Falls der Knoten y zu Z_s gehört und an der Stelle $M_{x,y}$ 1 steht, bleibt es auch 1.
   c) Falls der Knoten y zu Z_s nicht gehört und an der Stelle $M_{x,y}$ 0 steht, bleibt es auch 0.
   d) Falls der Knoten y zu Z_s nicht gehört, aber an der Stelle $M_{x,y}$ 1 steht, wird die Stelle $M_{x,y}$ zu 0.
2. Falls es sich um einen Knoten $x \in A \setminus F_s$ handelt, betrachten wir dazu die entsprechende Liste M_x und einen Knoten $y \in B$.
   a) Falls der Knoten y zu Z_s nicht gehört, aber an der Stelle $M_{x,y}$ 0 steht, bleibt es auch 0.
   b) Falls der Knoten y zu nicht Z_s gehört und an der Stelle $M_{x,y}$ 1 steht, bleibt es auch 1.
   c) Falls der Knoten y zu Z_s gehört, aber an der Stelle $M_{x,y}$ 1, wird die Stelle $M_{x,y}$ zu 0.
   d) Falls der Knoten y zu Z_s gehört und an der Stelle $M_{x,y}$ 0 steht, bleibt es auch 0.

Lemma 3. *Sei $C = (V_c, E_c) \subseteq G$ eine Zusammenhangskomponente. Sei $K = (F, Z)$ eine Spießkombiantion. Wir betrachten, was mit G nach der Verarbeitung von K passiert.*

(i) *Falls gilt: $F \cup Z = V_c$, dann entsteht keine neue Zusammenhangskomponente in G.*

(ii) *Falls gilt: $F \cup Z \subsetneq V_c$,dann entstehen zwei neue Zusammenhangskomponenten in G — C wird in zwei Komponenten gespalten.*

(iii) *Seien $C_1, C_2, ..., C_k \subset G$ voneinander unterschiedliche Zusammenhangskomponenten.*
Falls $F \cup Z$ aus mehreren Teilmengen aus $C_1, C_2, ..., C_k$ besteht, gelten für jede Komponente C_i ebenfalls (i) *und* (ii).

Beweis.
TODO: Beweis zu Ende

Die Beweise für die entsprechenden Punkte:

(i) Nach der Definition einer Zusammenhangskomponente gilt: $\nexists x \in V_c : x \in V \setminus V_c$. Bei Bearbeiteung von K werden nach Lemma 1 alle Kanten zwischen $x \in V \setminus V_c$ und $y \in V_c$ aus E entfernt. Deshalb werden bei so einer Spießkombination K keine Kanten entfernt.

(ii) Die beiden Mengen A und B gleichmächtig und ganz am Anfang ist G vollständig. Nach der ersten Spießkombination $K_1 = (F_1, Z_1)$, wobei $F_1 \neq A$, entstehen zwei Zusammenhangskomponenten: $C_1 \cup C_2 = G$. Dann sind $C_1 \cap A$ und $C_1 \cap B$ auch gleichmächtig. Ebenfalls sind $C_2 \cap A$ und $C_2 \cap B$ gleichmächtig. Nach 1 gilt, dass alle Kanten zwischen C_1 und C_2 aus E entfernt wurden, aber alle innerhalb von C_1 und innerhalb von C_2 beibehalten wurden. Dies bedeutet, dass die Komponenten C_1 und C_2 selbst vollständige, bipartite Graphen sind.

TODO: I.A. jest powyżej.
I.S.: co się dzieje po każdym K_i => własność grafu (pełność) pozostaje utrzymana na wszystkich spójnych

□

1.4 Zusammenhangskomponenten

Nach der Verarbeitung der allen m Spießkombinationen verfügen wir über den Graphen G, in dem viele Kanten in E entfernet wurden. Auf diese Weise können wir schon anfangen, die Indizes der Obstsorten aus W festzulegen. Definieren wir zunächst, was generell ein **Matching** ist.

Definition 2 (Matching)**.** *Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein ungerichteter Graph. Als ein **Matching** bezeichnen wir eine Teilmenge $\mathcal{S} \subseteq \mathcal{E}$, sodass für alle $v \in \mathcal{V}$ gilt, dass höchstens eine Kante aus $\mathcal{S}$ inzident zu v ist. Wir bezeichnen einen Knoten $v \in \mathcal{V}$ als in $\mathcal{S}$ **gematcht**, wenn eine Kante aus $\mathcal{S}$ inzident zu v ist.*

Zwischen verschiedenen Typen des Matchings unterscheidet man auch das **perfekte Matching**.

Definition 3 (Perfektes Matching)**.** *Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein ungerichteter Graph. Ein **perfektes Matching** ist so ein Matching, in dem alle Knoten aus $\mathcal{V}$ gematcht sind.*

Um die Aufgabe in der Form zu lösen, eignet sich gut der **Satz von Hall**, der als ein Ausgangspunkt der ganzen Matching–Theorie gilt. Um sich dieses Satzes zu bedienen, muss man noch den Begriff der **Nachbarschaft** einführen.

Definition 4 (Nachbarschaft)**.** *Sei $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ ein ungerichteter Graph. Für alle $X \subseteq \mathcal{V}$ definieren wir die **Nachbarschaft** von X als $N(X) = \{y \in \mathcal{V} \mid \forall x \in X : (x, y) \in \mathcal{E}\}$.*

Satz 1 (Satz von Hall)**.** *Sei $\mathcal{G} = (\mathcal{L} \cup \mathcal{R}, \mathcal{E})$ ein bipartiter, ungerichteter Graph. Es existiert ein perfektes Matching genau dann, wenn es für alle Teilmengen $\mathcal{K} \subseteq \mathcal{L}$ gilt: $|\mathcal{K}| \leqslant |N(\mathcal{K})|$.*

Beweis. Auf den Beweis verzichte ich. Ein Beweis ist beispielsweise hier [1] zu finden. □

An dieser Stelle ist es leicht, Folgendes festzustellen.

Lemma 4. *Seien $x \in A$ ein Knoten in G, seine Kardinalität $\Delta(x) = 1$, und der einzelne Nachbar von x sei $y \in B$. Dann gilt: $o(x, y)$.*

Beweis. Nach dem Satz von Hall ist für $x \in A$ die Bedingung $|x| = \Delta(x) = 1 \leqslant |N(x)| = 1$ erfüllt. Deshalb existiert ein perfektes Matching für $x \in A$ und das ist auch das einzelne mögliche Matching. Nach Axiom 1 hat jede Obstsorte genau einen einzigartigen Index, also ist der Index der Obstsorte x somit gefunden. □

Dennoch es bleibt noch der allgemeine Fall — für $\Delta(x) > 1$. Dazu stellen wir das Folgende fest.

Lemma 5. *Seien $x \in A$ ein Knoten in G und seine Kardinalität $\Delta(x) = k > 1$. Dann gehört x zu einer Zusammenhangskomponente $C = (V_c, E_c)$, wobei die Menge V_c aus insgesamt $2k$ Knoten $x_1, ..., x_k \in A$ und $y_1, ..., y_k \in B$ besteht. Für die Menge E_c gilt: $E_c = (A \setminus V_c) \times (B \setminus V_c)$. C ist deshalb selbt ein vollständiger, bipartiter Graph.*

Beweis. Der Beweis erfolgt durch Widerspruch.

TODO: Beweis

□

Nun kann man das folgende Lemma für den allgemeinen Fall formulieren.

Lemma 6. *Sei $C = (V_c, E_c)$ eine Zusammenhangskomponente in G. Dann existiert immer ein perfektes Matching zu C.*

Beweis. Nach Lemma 5 ist jede Zusammenhangskomponente ein vollständiger. bipartiter Graph. Nach Satz von Hall existiert ein perfektes Matching, wenn für alle Teilmengen $K \subseteq A \cap V_c$ gilt: $|K| \leqslant |N(K)|$. Die obere Behauptung kann für beliebig größe Mächtigkeiten $|K| = k \in \mathbb{N}$ durch die vollständige Induktion bewiesen werden.

Induktionsanfang: Für $k = 1$ hat der einzelne Knoten $x \in K \subseteq A \cap V_c$ die Kardinalität $\Delta(x) = 1$. Deshalb gilt: $|x| = 1 \leqslant |N(x)| = 1$, was in Lemma 4 schon bewiesen wurde. Damit stimmt die Behauptung für $k = 1$ und der Induktionsanfang ist erledigt.

[1] Anup Rao. Lecture 6 Hall's Theorem. October 17, 2011. University of Washington. [Zugang 21.01.2021] `https://homes.cs.washington.edu/~anuprao/pubs/CSE599sExtremal/lecture6.pdf`

Induktionsschritt: Es gelte die Aussage für ein $k \in \mathbb{N}$, also eine Teilemenge $K \subseteq A \cap V_c$ besteht aus k Knoten und jeder Knoten $x \in K$ hat die Kardinalität $\Delta(x) = k$. Es gelte also: $|K| \leqslant |N(K)|$.
Zu zeigen ist die Aussage für $k+1$, also für eine Teilmenge $K' \subseteq A \cap V_c$ der Mächtigkeit $|K'| = k+1$:

$$|K'| \leqslant |N(K')|.$$

Wir verifizieren:
Jeder Knoten in C hat den Grad $k+1$, also: $|K'| = k+1 \leqslant |N(K')| = (k+1)^2 = k^2 + 2k + 1$.
Folglich stimmt die Behauptung für $k+1$.

Der Induktionsschritt ist damit vollzogen und es wurde bewiesen, dass die Behauptung für beliebige Mächtigkeit von K gilt. Dadurch wurde auch bewiesen, dass es in einer Zusammenhangskomponente in G immer ein perfektes Matching gibt. □

Lemma 7. *Sei $C = (V_c, E_c)$ eine Zusammenhangskomponente in G. Wenn gilt: $\forall x \in A \cap V_c : x \in W$, dann werden alle $y \in B \cap V_c$ in W' hinzugefügt.*

Beweis. Nach Axiom 1 besitzt jede Obstsorte genau einen einzigartigen Index. Die Zusammenhangskomponente C beschreibt nach Lemmata 5 und 6, dass jede Obstsorte $x \in A \cap V_c$ jeden Index $y \in B \cap V_c$ haben kann, weil ein perfektes Matching stets existiert und C ein vollständiger, bipartiter Graph ist.
Dadurch, dass $\forall x \in A \cap V_c : x \in W$ gilt, ist ohne Bedeutung, welchen Index die jeweilige Obstsorte besitzt, da die Lösung des Problems eine Menge W' mit den Indizes der Obstsorten aus W sein soll. Dadurch, dass $A \cap V_c \subseteq W$ auch gilt: $B \cap V_c \subseteq W'$. □

Lemma 8. *Sei $C = (V_c, E_c)$ eine Zusammenhangskomponente in G. Wenn gilt: $\exists x \in A \cap V_c : x \notin W$, dann kann die Menge W' nicht eindeutig festgelegt werden.*

Beweis. Nach Axiom 1 besitzt jede Obstsorte genau einen einzigartigen Index. Die Zusammenhangskomponente C beschreibt nach Lemmata 5 und 6, dass jede Obstsorte $x \in A \cap V_c$ jeden Index $y \in B \cap V_c$ haben kann, weil ein perfektes Matching stets existiert und C ein vollständiger, bipartiter Graph ist.
Angenommen, $\exists p \in A \cap V_c \wedge p \notin W$. Dann ist es unmöglich, festzustellen, welcher Index aus $B \cap V_c$ der Obstsorte p gehört. Also ist es auch unmöglich, festzustellen, welche Indizes in W' hinzugefügt werden sollen. Deshalb, unabhängig von allen anderen Zusammenhangskomponenten des Graphen G, ist unmöglich, eine eindeutige Menge der Indizes der gewünschten Obstsorten festzulegen. Dadurch gibt es keine eindeutige Lösung zu diesem Problem für diese Eingabe. □

Lemma 9.
TODO: umschreiben

Wenn alle Zusammenhangskomponten, die die Wünsche aus W beinhalten, nur aus den Wüschen aus W bestehen, kann W' eindeutig und vollständig festgelegt werden.

Beweis. □

1.5 Prüfung auf Korrektheit der Eingabe

1.6 Laufzeit

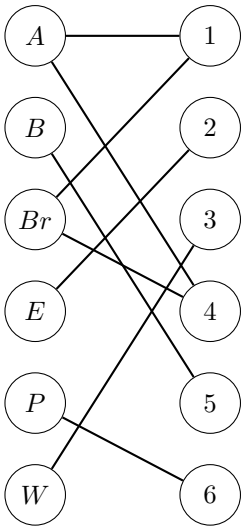
2 Umsetzung

2.1 Klasse `Solver`

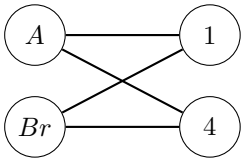
Die Matrix M wird als ein `vector` von `bitset` dargestellt. Dazu muss man erwähnen, dass ein `bitset` in C++ eine feste Länge besitzen muss. Dazu wurde die maximale Größe von n eingegeben, also 26. Das müsste ggf. im Program selbst umgestellt werden, falls man eine größere Datei einlesen möchte.
Die feste Länge ist auch der Grund dafür, dass die Bitmaske br im Teil 1.3 auf folgende Weise definiert wird:

$$br := \neg(bn) \wedge bf.$$

Im Fall, wenn man mit einer Bimaske einer festen Größe operiert, würde eine einfache Negation der Bitmaske bn nicht hinreichen.



(a) Der Graph nach der Analyse der allen Spießkombinationen



(b) Die ürbige Zusammenhangskomponente

Die Obstsorten werden als Strings eingelesen, aber in der Methode `readFile()` wird jeder Obstsorte ein interner Index[2] zugewiesen und in weiteren Operationen im Programm werden die Obstsorten als einfache Integers behandelt. Es werden dazu zwei Maps festgelegt: `fruit2ID` und `ID2Fruit`, in denen die Obstsorten und die entsprechenden internen Indizes gespeichert sind.

Jede Spießkombination wird als `pair<set<int>, set<int>`, also ein Tupel entsprechend aus der Menge der internen Indizes der Obstsorten und der Menge der Indizes aus der Aufgabenstellung. Alle Spießkombinationen werden in einem `vector` gespiechert, der `infos` heißt.

In `wishes` werden als ein `set` von Integers werden die internen Indizes der gewünschten Obstsorten, also der Elemente der Menge W, gespeichert. Analog werden in `result`, also ebenfalls ein `set` von Integers, die Indizes der gewünschten Obstsorten, also die Elemente der Menge W' hinzugefügt.

Der `vector`, der `used` heißt, wird verwendet, um die benutzten internen Indizes der Obstsorten, wie auch die Indizes der Obstsorten zu markieren, die im Graphen verwendet werden, da es in einzigen Textdateien ein größeres n gibt als die Mächtigkeit der entsprechenden Menge A. Diese Markierung zeigt sich bei der Prüfung auf korrekte Eingabe hilfreich.

Die Methode `analyzeInfo()` nimmt als Argument eine Spießkombination. In der Methode werden die drei Bitmasken bn, br, bf erstellt. Um die Laufzeit zu optimieren, wird durch die Menge der internen Indizes der Obstsorten `fruits` aus dieser Spießkombination gleichzeitig mit den Obstsorten aus der Matrix iteriert. Da die Menge `fruits` vorsortiert ist, müssen wir nicht bei jeder Obstsorte in der Matrix prüfen, ob sie sich in `fruits` befindet, um die Entscheidung zu treffen, welche der beiden Bitmasken anzuwenden. Nachdem alle m Spießkombinationen verarbeitet wurden, werden in der Methode `analyzeAllInfos()` alle übrigen 1–Beziehungen in der Adjazenzmatrix als Kanten in den Graphen `G` der Klasse `Graph` kopiert.

TODO: `checkCoherence()`

Nachdem die Korrektheit der Eingabe geprüft wurde, kann festgestellt werden, ob W' eindeutig bestimmt werden kann. Dazu dient die Methode `checkResult()`.

TODO: `checkResult()`

2.2 Klasse `Graph`

Diese Klasse ist grundsätzlich aus Übersichtlichkeits– sowie aus Vereinfachungsgründen entstanden. Theoretisch könnte man die enthaltenen Methoden in der Klasse `Solver` speichern.

[2] Nummerierung ab 0.

Der Graph ist als eine Adjazenzliste aus `vector` von `vector` von Integers gespeichert. Der Konstruktor nimmt zwei Parameter: die Größen der beiden Partitionen im bipartiten Grphen, obwohl sie in der Aufgabe gleich groß sind.

Zu den verfügbaren Methoden zählen: `addEdge()`, die eine ungerichtete Kante zwischen zwei Knoten einfügt; `deg()`, die die Kardinalität eines Knotens zurückgibt; `getNeighbors()`, die den `vector`, also die Adjazenzliste eines Knotens zurückgibt. Außerdem gibt es ein paar Methoden, die zum Debugging dienen.

3 Beispiele

3.1 Beispiel 0 (Aufgabenstellung)

Textdatei: `spiesse0.txt`

Apfel, Brombeere, Weintraube

1, 3, 4

3.2 Beispiel 1 (BWINF)

Textdatei: `spiesse1.txt`

Wünsche: Clementine, Erdbeere, Grapefruit, Himbeere, Johannisbeere

1, 2, 4, 5, 7

3.3 Beispiel 2 (BWINF)

Textdatei: `spiesse2.txt`

Wünsche: Apfel, Banane, Clementine, Himbeere, Kiwi, Litschi

1, 5, 6, 7, 10, 11

3.4 Beispiel 3 (BWINF)

Textdatei: `spiesse3.txt`

Wünsche: Clementine, Erdbeere, Feige, Himbeere, Ingwer, Kiwi, Litschi

unlösbar: Litschi gehört zur Komponente mit Grapefruit. Dabei ist Grapefruit kein Wunsch.

3.5 Beispiel 4 (BWINF)

Textdatei: `spiesse4.txt`

Wünsche: Apfel, Feige, Grapefruit, Ingwer, Kiwi, Nektarine, Orange, Pflaume

2, 6, 7, 8, 9, 12, 13, 14

3.6 Beispiel 5 (BWINF)

Textdatei: `spiesse5.txt`

Wünsche: Apfel, Banane, Clementine, Dattel, Grapefruit, Himbeere, Mango, Nektarine, Orange, Pflaume, Quitte, Sauerkirsche, Tamarinde

1, 2, 3, 4, 5, 6, 9, 10, 12, 14, 16, 19, 20

3.7 Beispiel 6 (BWINF)

Textdatei: `spiesse6.txt`

Wünsche: Clementine, Erdbeere, Himbeere, Orange, Quitte, Rosine, Ugli, Vogelbeere

4, 6, 7, 10, 11, 15, 18, 20

3.8 Beispiel 7 (BWINF)

Textdatei: `spiesse7.txt`

Wünsche: Apfel, Clementine, Dattel, Grapefruit, Mango, Sauerkirsche, Tamarinde, Ugli, Vogelbeere, Xenia, Yuzu, Zitrone

unlösbar: Apfel, Grapefruit und Xenia gehören zur Komponente mit Litschi. Dabei ist Litschi kein Wunsch. Ugli gehört zur Komponente mit Banane. Dabei ist Banane kein Wunsch.

3.9 Beispiel 8

Textdatei: `spiesse8.txt`

```
4
Dattel Apfel Banane
3
1 2 3
Apfel Banane Dattel
1 2
Apfel Banane
1 3
Clementine Dattel
```

Wünsche: Dattel, Apfel, Banane

Error: Es gibt Fehler in der Eingabedatei.

Die erste Spießkombination legt fest, dass Apfel, Banane und Dattel einen der folgenden Indizes besitzen: $\{1, 2, 3\}$. Das bedeutet auch, dass Clementine — die einzelne übrige Obstsorte — den Index 4 besitzt. Jedoch widerspricht dieser Zuweisung die dritte Spießkombination, laut der Clementine den Index 1 oder 3 hat.

4 Quellcode

./tex/spiesse.m