Aufgabe 2: Spießgesellen

Teilnahme-ID: 56860

Bearbeiter/-in dieser Aufgabe:
Christopher Besch

29. Dezember 2020

Inhaltsverzeichnis

1 Lösungsidee

Die von Donald beobachteten Spieße lassen sich wie folgt darstellen:

Schüsseln	Früchte
1, 4, 5	Apfel, Banane, Brombeere
3, 5, 6	Banane, Pflaume, Weintraube
1, 2, 4	Apfel, Brombeere, Erdbeere
2, 6	Erdbeere, Pflaume

Bestimmung der legalen Früchte Nun wird in einer Tabelle jede Schüssel mit den Früchten der Spieße, die diese Schüssel verwendeten, aufgelistet (eine Spalte pro Spieß). Zudem werden in einer weiteren Spalte alle Früchte notiert, die in Spießen vorkommen, die nicht diese Schüssel verwenden, sie werden verbotene Früchte genannt. Diese Früchte können nicht von dieser Schüssel sein.

Schüssel	Spieße von dieser Schüssel		Verbotene Früchte
1	Apfel, Banane, Brombeere	Apfel, Brombeere, Erdbeere	Banane, Weintraube, Erdbeere, Pflaume
2	Apfel, Brombeere, Erdbeere	Erdbeere, Pflaume	Apfel, Brombeere, Banane, Weintraube, Pflaume
3	Banane, Pflaume, Weintraube		Banane, Apfel, Brombeere, Erdbeere, Pflaume
4	Apfel, Banane, Brombeere	Apfel, Brombeere, Erdbeere	Banane, Weintraube, Erdbeere, Pflaume
5	Apfel, Banane, Brombeere	Banane, Pflaume, Weintraube	Apfel, Brombeere, Erdbeere, Pflaume
6	Banane, Pflaume, Weintraube	Erdbeere, Pflaume	Banane, Apfel, Brombeere, Erdbeere

Nun werden für jede Schüssel alle möglicherweise vorliegenden Früchte gesucht, diese werden im Folgenden legale Früchte genannt. An jede Schüssel liegt eine Frucht aus der Menge der legalen Früchte für diese Schüssel.

Alle legalen Früchte für eine bestimmte Schüssel müssen zwei Voraussetzungen erfüllen:

1. Die Frucht muss in allen Spießen vorkommen, die diese Schüssel benutzen.
2. Die Frucht darf für diese Schüssel nicht verboten sein.

Alle legalen Früchte werden nun markiert:

Schüssel	**Spieße von dieser Schüssel**		**Verbotene Früchte**
1	Apfel, Banane, Brombeere	Apfel, Brombeere, Erdbeere	Banane, Weintraube, Erdbeere, Pflaume
2	Apfel, Brombeere, Erdbeere	Erdbeere, Pflaume	Apfel, Brombeere, Banane, Weintraube, Pflaume
3	Banane, Pflaume, Weintraube		Banane, Apfel, Brombeere, Erdbeere, Pflaume
4	Apfel, Banane, Brombeere	Apfel, Brombeere, Erdbeere	Banane, Weintraube, Erdbeere, Pflaume
5	Apfel, Banane, Brombeere	Banane, Pflaume, Weintraube	Apfel, Brombeere, Erdbeere, Pflaume
6	Banane, Pflaume, Weintraube	Erdbeere, Pflaume	Banane, Apfel, Brombeere, Erdbeere

Auswahl der Schalen für Donald Die folgenden Früchte sind (von Donald) gefordert: Weintraube, Brombeere und Apfel

Es werden alle Schüsseln durchgegangen.

- Wenn alle legalen Früchte einer Schüssel gefordert sind, wird diese Schüssel ausgewählt, da ganz egal, welche legale Frucht die tatsächlich an der Schüssel liegende ist, die Schüssel eine geforderte Frucht liefert.
- Wenn keine der legalen Früchte gefordert sind, ist dies kein ausgewählter Schüssel.
- Wenn einige, aber nicht alle legalen Früchte gefordert sind, ist nicht genügend Information vorhanden, da nicht gesagt werden kann, ob der Schüssel eine ausgewählte Frucht enthält oder nicht.

Es ergibt sich die Menge aller ausgewählten Schüsseln: 1, 3 und 4

Die Schüsseln 1 und 2 liefern jeweils entweder Äpfel oder Brombeeren. Da beide Früchte gefordert sind, ist diese Ungewissheit irrelevant.

Mit dieser Methode kann in allen Fällen eine Aussage gefällt werden, ob genügend Information vorliegt und wenn dies der Fall ist, die Menge der ausgewählten Schüsseln angegeben werden.

2 Umsetzung

Die Lösungsidee wird in C++ implementiert.

Einlese der Eingabedatei Als erster Schritt wird in der Funktion **read_file** die Eingabedatei gelesen. Hierbei wird überprüft, ob die Eingabedatei dem gegebenen Format entspricht, wenn nicht wird das Programm abgebrochen. Hierzu wird ein Makro **raise_error()** verwendet, dass die Ausführung des Programmes abbricht und eine möglichst informative Fehlermeldung zurückgibt. Dieses Makro wird ebenfalls für alle Methoden aller Klassen verwendet, um z.b. Segmentation Faults zu verhindern.

Die Fruchtnamen werden mithilfe einer Instanz der Klasse **LookupTable** aufsteigend und anfangend bei 0 in Glanzzahlen, Frucht IDs, übersetzt. Dadurch ist die Anzahl an verschieden Früchten nicht limitiert. Ein weiterer Vorteil ist, dass da die ID aufsteigend und bei 0 anfangend verteilt werden, die IDs als Indizes in Bit Fields verwendet werden können (hierzu wird std::vctor<bool> verwendet, da die Anzahl der Früchte bei compile time nicht bekannt ist). So liegt für jede Frucht ein Bit/Bool vor, ist er *true*, ist diese Frucht enthalten, ist sie $false$, ist sie nicht enthalten. Diese kompakte Datenstruktur ist

ressourcenschonender im Vergleich zu einem std::vector<std::string> mit den Namen aller enthaltenden Früchten.

Die Namen für die Schüsseln werden auf die gleiche Weise behandelt. Zwar ist angegeben, dass alle Schüsseln nummeriert sind, es ist allerdings nicht klar, wie die Nummern verteilt werden, daher ergibt es Sinn, die Namen der Schüsseln als Strings zu behandeln und ebenfalls eine Instanz von **LookupTable** zu verwenden. So können auch für die Schüsseln Bit Fields verwendet werden. Ein zusätzlicher Vorteil ist, dass so jegliche Namen für die Schüsseln erlaubt sind.

Als Größe für die Bit Fields wird die angegebene Anzahl an Früchten verwendet.

Es werden Instanzen der Klasse **Skewer** erstellt, diese enthalten die Menge der Früchte auf einem Spieß und die Menge der Schüsseln, von denen die Früchte genommen sind. Hierbei werden, wie bereits erwähnt, Bit Fields verwendet. An dieser Stelle werden fehlende Schüsseln und fehlende Früchte hinzugefügt, sodass die Anzahl an Schüsseln und Früchten der angegebenen Anzahl an Früchten entspricht. Diese neuen Früchte und Schüsseln erhalten Namen wie „new_bowl0“ oder „new_fruit0“. Es wird überprüft, ob bereits eine Frucht mit dem Namen „new_fruit0“ vorliegt, ist dies der Fall wird „new_fruit1“ versucht usw. Gleiches gilt für neue Schüsselnamen.

Bestimmung der Legalen Früchte In der Funktion **determine_legal_fruits** werden die legalen Früchte bestimmt, wie bereits in der Lösungsidee beschriebene.

Zuerst werden dazu für jede Schale **Bowl** Instanzen erstellt. Diese enthalten **allowed_fruit_sets**, ein std::vector mit einem Bit Field pro Spieß, der diese Schale verwendet, alle verbotenen Früchte (**disallowed_fruits**), alle legalen Früchte (**legal_fruits**) und zwei bools, **selected** und **possible_selected**, auf die später eingegangen wird. Es werden alle Spieße durchgegangen, wenn der Spieß diese Schale verwendet, werden dessen Früchte zu **allowed_fruit_sets** hinzugefügt, wenn nicht zu **disallowed_fruits**.

Nun werden die legalen Früchte für jede Schale bestimmt. Dafür werden für jede Frucht die beiden in der Lösungsidee genannten Voraussetzungen überprüft.

Auswahl der Schalen Im letzten Schritt werden alle Schalen durchgegangen. Hierbei wird überprüft, wie viele der legalen Früchte einer Schale gefordert sind. Sind alle gefordert wird **selected** auf *true* gesetzt, sind einige aber nicht alle gefordert, wird **possible_selected** auf *true* und **possible** auf *false* gesetzt.

Wenn **possible** nach Durchlauf aller Schalen *true* bleibt, ist genügend Information vorhanden, um eine Aussage treffen zu können.

main In der main Funktion werden die drei genannten Funktionen ausgeführt und die Ergebnisse in der Konsole ausgegeben.

3 Beispiele

Nun wird das Programm mit allen Beispieldateien ausgeführt.

spiesse0.txt Dies ist das Beispiel, das in dem Aufgabenblatt vorgestellt ist.

Das Programm liefert die richtige Antwort, es ist genügend Information vorhanden und die ausgewählten Schalen sind 1, 4 und 3.

spiesse1.txt Das Programm erkennt, dass genügend Information vorliegt und wählt die Schalen 2, 1, 4, 5 und new_bowl0 aus.

Hier lässt sich erkennen, dass eine Schale in keinem Spieß vorkommt und einen Namen zugeteilt bekommen hat.

spiesse2.txt Erneut wird das korrekte Ergebnis produziert, es liegt genügend Information vor und die Schalen 6, 1, 7, 11, 5 und 10 wurden ausgewählt.

spiesse3.txt In diesem Fall liegt nicht genügend Information vor, um eindeutig sagen zu können, welche Schalen auszuwählen sind. Dies gibt das Programm korrekt aus und gibt and, dass die Schalen 7, 10, 12, 5, 1 und 8 definitiv geforderte Früchte enthalten und dass entweder die Schale 2 oder 11 eine geforderte Frucht enthält, es liegt allerdings nicht genügend Information vor, um genau bestimmen zu können, welche das ist.

In der ersten Zeile der Beispieldatei wird die Anzahl der Früchte angegeben, in diesem Fall 15. Allerdings sind nur 14 verschiedene Früchte in den Spießen und den geforderten Früchten vorzufinden. Daher wird die fehlende Frucht new_fruit0 genannt, auch wenn sie für die Lösung des Problems irrelevant ist, da sie nicht gefordert wurde.

spiesse4.txt Erneut können die Schalen mit den geforderten Früchten genau bestimmt werden, 7, 9, 2, 6, 8, 12, 13 und 14.

spiesse5.txt Genau wie bei **spiesse3.txt** ist die angegebene Anzahl an Früchten zu hoch, weshalb eine Frucht new_fruit0 genannt wird. Das Programm liefert die korrekte Ausgabe, es gibt eine Lösung und die ausgewählten Schalen sind 16, 14, 6, 9, 2, 3, 1, 19, 5, 10, 12, 4 und 20.

spiesse6.txt Es liegt genügend Information vor und die ausgewählten Schalen sind 18, 15, 20, 11, 4, 7, 10 und 6.

spiesse7.txt Es liegt nicht genügend Information vor und die Schalen 14, 8, 5, 24, 23, new_bowl0, new_bowl1 und new_bowl2 liefern garantiert geforderte Früchte, wobei new_bowl0, new_bowl1 und new_bowl2 Schalen sind, die für keinen der angegebenen Spieße verwendet wurden sind. Zusätzlich gibt das Programm aus, dass vier der folgenden Schalen geforderte Früchte enthalten: 3, 20, 26, 10, 25 und 18

3.1 Eigene Beispiele

myspiesse0.txt In der Aufgabenstellung ist angegeben, dass die Namen der Früchte immer unterschiedlichen Anfangsbuchstaben haben. Dieses Programm limitiert, die Eingabedatei allerdings nicht auf diese Weise.

Dies ist in diesem Beispiel gezeigt, es ist eine Kopie von spiesse0.txt, wobei alle Früchte den selben Anfangsbuchstaben haben. Das Programm produziert das korrekte Ergebnis, Schüsseln 1, 4 und 3.

myspiesse1.txt Es handelt sich erneut um eine Kopie von spiesse0.txt. Hier ist die angegebene Anzahl an Früchten deutlich höher (10) als die in den Spießen und geforderten Früchten gelisteten Schalen und Früchten (6). Das Programm erstellt daher vier neue Früchte und Schalen. Es produziert das korrekte Ergebnis: 1, 4 und 3

myspiesse2.txt Die Datei sieht wie folgt aus:

1
Weintraube
0

Es ist nur eine Frucht vorhanden, die ebenfalls gefordert wird. Es liegt keinerlei Information über jegliche Spieße vor.

Das Programm produziert das korrekte Ergebnis, new_bowl0, da nur eine Schale vorliegt und diese damit die geforderte Frucht enthalten muss.

myspiesse3.txt Die Datei sieht wie folgt aus:

2
Weintraube

0 Der einzige Unterschied zu dem vorherigen Beispiel ist die Anzahl aller Früchte total. In Diesem Fall sind zwei Früchte vorhanden.

Das Programm erstellt eine neue Frucht, da nur eine gefunden werden kann, Weintraube, und zwei neue Schalen. In diesem Fall ist nicht klar, in welcher Schale die geforderten Weintrauben liegen. Dies gibt das Programm korrekt aus.

myspiesse4.txt Dies ist eine leere Datei, weshalb das Programm mit einer Fehlermeldung abbricht: „File Parsing Error: More lines required!“

myspiesse5.txt Dies ist eine Kopie von spiesse7.txt, wobei die Nummern aller Schalen durch zufällige Wörter ersetzt wurden. Das Programm produziert problemlos das korrekte Ergebnis.

myspiesse6.txt In diesem Fall wird für die Anzahl aller Früchte „fdfhasdfgdfhsdfh“ eingesetzt. Das Programm bricht mit einem Fehler ab, „File Parsing Error: Can’t convert "fdfhasdfgdfhsdfh"to int!“

4 Quellcode

Dies ist der Teil der Funktion **determine_legal_fruits**, der die legalen Früchte für alle Schüsseln auswählt. „fruit_look_up.get_amount()“ gibt die Anzahl aller Früchte zurück.

```
for (Bowl &bowl : bowls)
{
    for (int fruit_idx = 0; fruit_idx < fruit_look_up.get_amount(); fruit_idx++)
    {
        // also true when no skewers given <- no info means everything is possible
        bool in_all = true;
        for (const std::vector<bool> &fruit_set : bowl.get_allowed_fruit_sets())
            if (!fruit_set[fruit_idx])
            {
                in_all = false;
                break;
            }

        // for this to be a legal fruit, it has to be in every allowed fruit set
        // and not a disallowed fruit
        if (!bowl.get_disallowed_fruits()[fruit_idx] && in_all)
            bowl.add_legal_fruit(fruit_idx);
    }
}
```

Die Funktion **get_selected_bowls** verwendet den folgenden Teil, um die jeweils legalen Früchte für alle Schalen auszuwählen.

```
possible = true;
for (Bowl &bowl : bowls)
{
    bool all_legal_are_requested = true;
    bool all_legal_are_not_requested = true;
    // go through all the fruits, one of which this bowl provides
    for (int idx = 0; idx < fruit_look_up.get_amount(); idx++)
    {
        if (bowl.get_legal_fruits()[idx])
        {
            if (requested_fruits[idx])
                all_legal_are_not_requested = false;
            else
                all_legal_are_requested = false;
        }
    }

    // when some of this bowls legal fruits are required and others aren't,
    // there isn't enough information
    if (!all_legal_are_requested && !all_legal_are_not_requested)
    {
        possible = false;
        bowl.set_possibly_selected();
    }
    else if (all_legal_are_requested)
        bowl.set_selected();
}
```