33. Bundeswettbewerb Informatik

Buffet-Lotterie (Aufgabe 3)

Johannes Heinrich, Marian Dietz

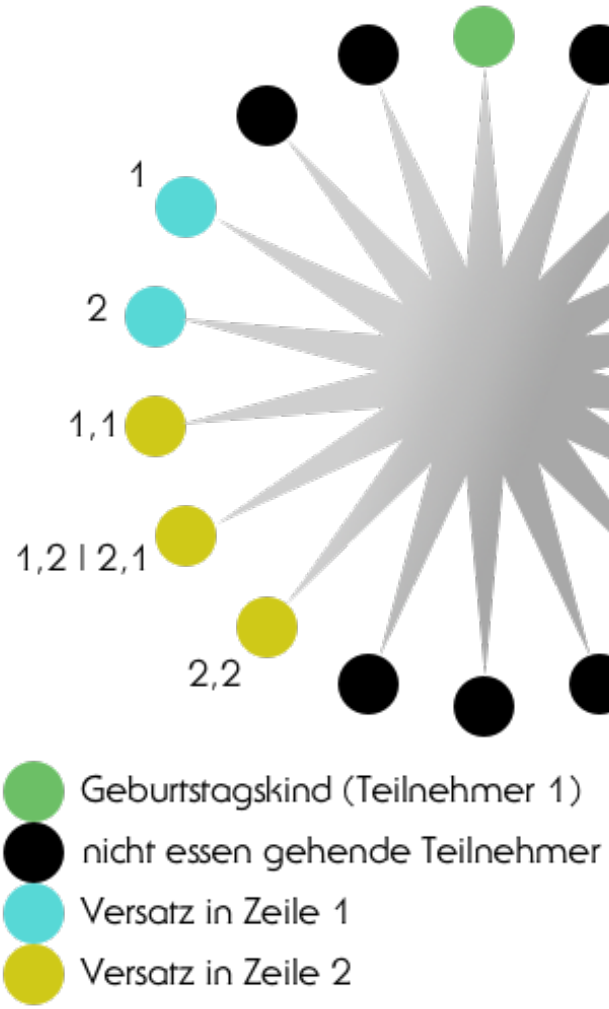
Lösungsidee

Der Algorithmus zur Berechnung der schnellstmöglichen Variante kann tabellenartig dargestellt werden. Jede Zeile steht für eine „Runde“, also bis der Vers so oft abgezählt wurde, dass er beim nächsten Durchgang über das Geburtstagskind hinaus gehen würde. Dies soll nun an der folgenden Tabelle erklärt werden, wobei 18 Personen an der „Buffet-Lotterie“ teilnehmen.

Nummer	Alte Anzahl der Gesamtteilnehmer	Anzahl der Teilnehmer, die essen gehen	Versatz
1	18	1	3 I 4
2	17	1	4 I 5 I 6
3	16	1	4 I 5 I 6 I 7
4	15	1	3 I 4 I 5 I 6 I 7
5	14	1	1 I 2 I 3 I 4 I 5 I 6
6	13	1	11 I 12 I 13 I 14 I 15 I 16 I 17

Zunächst hat jede Zeile eine eindeutige Nummer k, welche in jeder Zeile um eins erhöht wird und bei eins beginnt. Diese Variable spielt jedoch keine große Rolle und wird nur in dieser Tabelle verwendet, um im Folgenden jede Zeile eindeutig bestimmen zu können.
Als Nächstes findet in der Tabelle die alte Anzahl der Gesamtteilnehmer Platz. Diese Variable nennen wir a_k. Das Geburtstagskind ist immer die erste Teilnehmerin, sie nennen wir also *Teilnehmerin 1*.
Danach kommt die Anzahl der Teilnehmer, welche essen gehen (e_k).
Wichtig dabei ist, dass a_k die alte Anzahl der Teilnehmer ist. Deshalb wirkt sich e_k erst auf die nächste Zeile aus. Würden beispielsweise in einem Durchgang zwei Teilnehmer essen gehen, wäre a_k erst in der Zeile danach um zwei geringer: $a_{k+1} = a_k - 2$. Bei diesem Beispiel spielt das jedoch keine Rolle, da e_k stets eins ist.
Um nun herauszufinden, wie viele Silben das Geburtstagskind in jeder Reihe sagen soll, wird der Versatz v mit der Länge l berechnet. v gibt an, wie viele Teilnehmer die letzte Silbe vom Geburtstagskind entfernt ist. Wenn also $a = 17$ und $v = 4$, wurde die letzte Silbe vom *Teilnehmer 14* gesagt, *denn Teilnehmer 14* ist 4 Teilnehmer vom Geburtstagskind (*Teilnehmer 1*) entfernt.
Die Länge des Versatzes besagt, wie viele Versätze möglich sind. Dabei ist der „normale“ Versatz v stets der kleinste. Der größte Versatz ergibt sich aus der Länge des Versatzes und dem normalen Versatz.

Ist beispielsweise der Versatz 3 und die Länge des Versatzes ist 2, dann kann der Versatz 3 oder 4 sein. Ist der Versatz 4 und die Länge des Versatzes ist 3, dann kann der Versatz 4, 5 oder 6 sein. Diese verschiedenen Versätze kommen dadurch zustande, dass *Teilnehmer 1* immer die Wahl zwischen einer und zwei Silben hat,



Diese Grafik veranschaulicht die ersten beiden Zeilen der Tabelle mit 18 Teilnehmern. Dabei sieht es so aus, als ob der Versatz der zweiten Zeile fünf statt vier ist. Dies liegt darin begründet, dass nach der ersten Zeile ein hier blau gefärbter Teilnehmer essen geht. Deshalb muss man sich beim Betrachten einen blauen Teilnehmer wegdenken, wodurch der Versatz wieder zu 4 wird.
Wie viele Silben das Geburtstagskind jede Runde sagt, steht neben den Teilnehmern, welche die letzte Silbe sagen. Wenn das Geburtstagskind am Anfang eine Silbe sagt, ist der Versatz 3 (der obere blauer Punkt). Bei zwei Silben wird der Versatz 4 (der untere blaue Punkt). Aus diesem Grund ist die Länge des Versatzes in der ersten Zeile zwei: $l_1 = 2$.
Wenn das Geburtstagskind in der ersten Runde eine und in der zweiten Runde wieder nur eine Silbe sagt, ist der Versatz logischerweise 4, die letzte Silbe sagt dann der Teilnehmer des oberen gelb gefärbtem Punktes. Sagt das Geburtstagskind zwei mal zwei Silben, ist der Versatz sechs. Der letzte Teilnehmer ist dann der untere gelbe Punkt. Nun gibt es noch eine dritte Möglichkeit. Dabei ist der Versatz in der zweiten Runde fünf. Das Geburtstagskind sagt dann in einer Runde eine Silbe und in der anderen Runde zwei Silben. Dabei ist es egal, in welcher Reihenfolge dies geschieht. Es spielt immer nur eine Rolle, wie oft das Geburtstagskind wie viele Silben sagt. Dafür

gibt es eine einfache Begründung: es gehen immer gleich viele Teilnehmer essen, da gleich viele Silben insgesamt gesagt werden. Wer diese Teilnehmer sind, ist im Endeffekt völlig egal, da es lediglich darum geht, dass das Geburtstagskind möglichst schnell dran ist. Ab der zweiten Zeile steigt die Länge des Versatzes also immer um eins.

Nun geht es darum, wie die Tabelle berechnet wird. Zunächst wird die alte Anzahl der Teilnehmer errechnet. In der ersten Zeile ist dies logischerweise die Gesamtzahl der Teilnehmer, also wie viele insgesamt am Anfang mitspielen: $a_1 = a$. Für $k \neq 1$ gilt $a_k = a_{k-1} - e_{k-1}$. Hier wird einfach aus der Zeile davor die alte Anzahl der Teilnehmer genommen und davon wird die Anzahl der Teilnehmer, welche essen gehen, abgezogen. Dadurch erhält man die Anzahl der Teilnehmer, welche jetzt noch im Kreis stehen.

Danach wird berechnet, wie viele Teilnehmer in diesem Durchgang essen gehen. Dafür nehmen wir die folgende Formel: $e_k = \frac{h_k}{16}$. h_k ist die Anzahl der Teilnehmer, welche bis zum Ende dieser Runde eine Silbe sagen. Diese Zahl wird durch 16 geteilt, da der Abzählreim insgesamt 16 Silben beinhaltet. Durch die oben genannte Formel wird dann ausgerechnet, wie oft der Abzählreim in der Runde k vollständig aufgesagt wird, also „wie oft die 16 in h_k reinpasst“. Sollte dies keine ganze Zahl sein, wird alles nach dem Komma „abgeschnitten“, da e_k eine ganze Zahl sein muss. Es können nämlich nur ganze Personen essen gehen.
$h_1 = a_1$, da in der ersten Runde lediglich so viele Teilnehmer eine Silbe sagen können, wie es auch Teilnehmer gibt. Wenn $k \neq 1$, gilt folgende Gleichung: $h_k = v_{k-1} + a_k$. Hier wird zu der Anzahl der Teilnehmer noch der Versatz der letzten Zeile addiert, da die Teilnehmer, welche „nach dem Versatz“ kommen würden, auch noch in dieser Rechnung Platz finden müssen. Diese Teilnehmer sagen schließlich auch noch je eine Silbe.

Abschließend wird v_k berechnet. Die Formel dafür lautet $v_k = a_k - (16 \cdot e_k - v_{k-1})$, wobei $v_0 = 1$.
$16 \cdot e_k$ ist die Anzahl der Silben, welche insgesamt in dieser Runde gesagt werden, da e_k Personen essen gehen und für jede Person, die essen geht, 16 Silben gesagt werden müssen. Davon wird der „Überstand“, also der Versatz der letzten Runde subtrahiert, da die letzte Runde bis zum Geburtstagskind noch zu Ende geführt werden muss. Diese Differenz subtrahieren wir schließlich von der Anzahl der Teilnehmer, die noch im Kreis stehen. Damit wissen wir dann, wie viele Teilnehmer Abstand noch zum Geburtstagskind sind.
Für die erste Zeile setzen wir den Versatz der vorherigen, nicht existierenden Zeile 0 auf 1, denn das Geburtstagskind sagt die erste Silbe. Aus diesem Grund können wir der Einfachheit halber annehmen, dass der letzte Teilnehmer vor dem Geburtstagskind auch die letzte Silbe davor gesagt hat, wodurch der Versatz eins wäre.

Jedoch kann es auftreten, dass e_k null ist. Das passiert immer dann, wenn h_k kleiner als 16 ist, wie es in Zeile 6 der Tabelle oben geschehen ist. In einem solchen Fall wird e_k einfach auf eins gesetzt, da mindestens eine Person essen gehen soll. Jedoch kann dadurch ein weiterer Spezialfall auftreten. Folgende Gleichung gehört zu Zeile 6:
$v_6 = a_6 - (16 \cdot e_6 - v_5) = 13 - (16 \cdot 1 - 1) = -2$. Nach dieser Rechnung wäre der Versatz negativ, also auf der anderen Seite des Geburtstagskindes. Außerdem bedeutet das, dass wir Teilnehmer 1 „überrundet“ haben, das Geburtstagskind war also mehrmals an der Reihe. Wenn der Versatz negativ oder gleich null ist, ändern wir demnach zwei Dinge:

1. Zum Versatz wird die Anzahl der Teilnehmer addiert. Damit bewirken wir, dass der Versatz wieder von der „richtigen“ Seite gezählt wird, da wir sozusagen eine Runde hinzufügen müssen, wie bereits unter Punkt eins beschrieben. Um das Beispiel erneut aufzugreifen: zum ursprünglichen Versatz -2 wird 13 addiert, womit der Versatz bei 11 liegt.
2. Wenn der letzte Versatz auch das Geburtstagskind „überrundet“, dann wird die Länge des Versatzes erneut um eins erhöht, da das Geburtstagskind noch einmal an der Reihe war und dadurch eine Entscheidung mehr treffen konnte. Hier ist es wichtig, zwischen dem „normalen“ und dem letzten Versatz zu unterscheiden. Punkt 1 tritt *immer* in Kraft, wenn der neu berechnete Versatz negativ oder gleich null ist. Der letzte Versatz tritt jedoch immer dann in Kraft, wenn das Geburtstagskind vorher bereits immer zwei Silben gesagt hat. Die Versatzlänge wird also immer nur dann erneut um eins erhöht, wenn das Geburtstagskind noch einmal zwei Silben sagt. Das Geburtstagskind kann aber beim letzten Versatz nicht erneut zwei Silben sagen, wenn der letzte Versatz nicht über das Geburtstagskind hinausragt. Wenn das Geburtstagskind bei einem der anderen Versätze zwei Silben sagt, kann dies ignoriert werden, da der Versatz in einem solchen Fall zwar um eins größer werden würde, der dadurch enstehende Versatz ist aber schon im spätestens letzten Versatz enthalten, da das Geburtstagskind dort bereits mindestens einmal eine Silbe mehr gesagt hat.
   Deswegen wird die Versatzlänge nur dann erneut inkrementiert, wenn auch der letzte Versatz das Geburtstagskind beim aktuellen Durchgang überragt.

Bei sehr kleinen Teilnehmerzahlen kann es passieren, dass der Versatz danach immer noch im negativen Bereich ist. Dann wurde das Geburtstagskind erneut überrundet und die beiden Schritte werden so oft wiederholt, bis der Versatz größer als null ist.

Das Einzige, was jetzt noch fehlt, ist die Abbruchbedingung und das Berechnen der Lösung. Wir wissen, dass wir ein Ziel erreicht haben, wenn eine der Zahlen des Versatzes entweder 15, 16 oder die Anzahl der Teilnehmer in der letzten Zeile entspricht. Wenn also eine dieser Zahlen in der Spalte mit dem Versatz in der Tabelle auftritt, wird abgebrochen. In der Beispielstabelle oben ist dies in Zeile 6. Dort stehen zwar gleich alle drei Abbruchbedingungen (in diesem Fall 13, 15 und 16), aber die erste, die dort auftaucht, wird verwendet, um die Lösung zu generieren. In diesem Fall wäre das die 13, also die Anzahl der Teilnehmer in Zeile 6.
Das Ziel wurde auch erreicht, wenn zum Versatz die alte Anzahl der Teilnehmer gehört, da dies bedeutet, dass der Versatz eine Runde über *Teilnehmer 1* „ragt“. Deswegen könnte man den Versatz der alten Anzahl der Teilnehmer auch null nennen. Da jedoch festgelegt wurde, dass der Versatz mit steigender Versatzlänge gleichmäßig steigt, bleiben wir dabei, dass dieser Versatz der alten Anzahl entspricht.
Wie die Lösung errechnet wird, wird wieder am folgenden Beispiel erklärt: „11 | 12 | 13 | 14 | 15 | 16 | 17“. Dies ist der komplette Versatz der Zeile 6 der Beispielstabelle. Die 13 soll als Abbruchbedingung dienen. Der Versatz ganz rechts (17) gilt nur dann, wenn immer zwei Silben gesagt wurden. Der Versatz ganz links (11) gilt nur dann, wenn immer eine Silbe gesagt wurde. Daher muss so oft eine Silbe gesagt werden, bis der Versatz auf 11-13 eingeschränkt ist. Dies geschieht dadurch, dass das Geburtstagskind 5 Mal nur eine Silbe sagt, denn bei jedem Mal wird der Versatz um eins kleiner. Danach sagt das Geburtstagskind noch 2 Mal 2 Silben, damit der Versatz von links aus eingeschränkt wird, sodass der Versatz *genau* 13 ist und es keinen anderen Wert mehr gibt. Daraus folgt schließlich folgende Formel: $s_1 = v_6 + l_6 - a_6 - 1$. Diese Formel dient zur Berechnung, wie oft das Geburtstagskind eine Silbe sagt. Der letzte Versatz (17) ist $v_6 + l_6 - 1$, da zum Versatz v_6 die Länge des Versatzes l_6 addiert wird und vom Ergebnis noch

eins subtrahiert werden muss, da die Versatzlänge den ursprünglichen Versatz 11 bereits mit einschließt. Davon wird nun a_6 abgezogen, damit am Ende der Abstand zwischen a_6 und dem letzten möglichen Versatz herauskommt. Durch $s_2 = l_6 - s_1 - 1$ wird errechnet, wie oft das Geburtstagskind zwei Silben sagen soll. Dabei wird von der Versatzlänge subtrahiert, wie oft eine Silbe gesagt wird, da der Versatz dadurch bereits vorher eingeschränkt wurde. Danach wird noch davon noch eins subtrahiert, da zum Schluss eine Zahl, in diesem Fall die 13 noch übrig bleiben soll.
Etwas anders wird die Lösung generiert, wenn die 15 oder die 16 als Bezugspunkt dient. Für 15 wird die folgende Gleichung verwendet: $s_1 = v_6 + l_6 - 1 - 15 = v_6 + l_6 - 16$, die Gleichung für 16 lautet $s_1 = v_6 + l_6 - 1 - 16 = v_6 + l_6 - 17$. Dabei dient $v_6 + l_6 - 1$ wieder dafür, den letzten Versatz ausfindig zu machen. Davon subtrahiert wird diesmal nicht die Anzahl der Teilnehmer, sondern 15 bzw. 16. Diese beiden Gleichungen werden wieder verwendet, um herauszufinden, wie oft das Geburtstagskind nur eine Silbe sagen soll. $s_2 = l_6 - s_1 - 1$ dient wieder als Berechnung dafür, wie oft *Teilnehmer 1* zwei Silben sagen soll. Diese Formeln schränken den Versatz wieder so ein, dass nur 15 bzw. 16 am Ende übrig bleibt. Jedoch muss dabei noch eine abschließende Sache gemacht werden:
Wenn der Versatz 16 ist, dann hat das Geburtstagskind die letzte Silbe noch nicht gesagt. Der Vers muss noch ein weiteres Mal komplett fertig gesprochen werden, damit *Teilnehmer 1* auch die letzte Silbe sagt. Also muss s_1 noch um eins hochgezählt werden.
Ähnlich ist es bei einem Versatz von 15. Der Vers muss auch hier ein weiteres Mal gesprochen werden, jedoch muss *Teilnehmer 1* zum Schluss *zwei* Silben sagen, damit das Geburtstagskind auch als letztes dran ist und essen gehen kann. Deswegen wird hier s_2 um eins hochgezählt.

Umsetzung

Das Programm wurde objektorientiert in der Programmiersprache Java geschrieben. Ausgeführt werden kann das Programm in einer Konsole / einem Terminal mit dem Befehl „java -jar [Pfad zur .jar-Datei]“. Es gibt folgende Klasse:

- `Row` - Steht für eine Zeile in der Tabelle und besitzt deren Eigenschaften. Diese Klasse kann überprüfen, ob die Zeile eine Lösung darstellt und kann - wenn dies der Fall ist - die Lösung in die Konsole ausgeben.
- `BuffetLotterieGenerator` - Generiert die Lösung.
- `BuffetLotterie` - Hauptklasse, liest Anzahl der Teilnehmer ein, startet `BuffetLotterieGenerator` und lässt `Row` die Lösung in die Konsole ausgeben.

Nachdem `BuffetLotterie` die Anzahl der Teilnehmer eingelesen hat, wird diese einer neuen Instanz von `BuffetLotterieGenerator` direkt im Konstruktor übergeben. Direkt danach wird die Methode `generate()` dieser Klasse ausgeführt, in welcher die Lösung generiert wird.

Zunächst wird eine neue Variable `row` erstellt, welcher noch kein Wert zugewiesen wird. Eine `while`-Schleife wird solange wiederholt, bis die zuletzt erstellte Zeile eine Lösung ist. Ist dies irgendwann der Fall, wird die Zeile zurückgegeben und `BuffetLotterie` ruft `Row.printSolution()` auf.

Innerhalb der Schleife passiert Folgendes: als Erstes wird die alte Anzahl der Teilnehmer a_k berechnet. Danach wird die Methode `e()` aufgerufen, welche die Anzahl der Teilnehmer, welche essen gehen können, ausrechnet und den Wert zurückgibt. Danach werden der normale und der letzte Versatz durch `versatz()` und `letzterVersatz()` generiert und die Versatzlänge wird berechnet. Durch eine weitere `while`-Schleife immer wieder überprüft, ob der Versatz kleiner oder gleich null ist, wodurch Versatz und Versatzlänge eventuell geändert werden. Danach wird die Variable `row` auf eine mit diesen Werten neu erstellte Instanz gesetzt, um kurze Zeit später wieder im Kopf der Schleife auf eine Lösung überprüft zu werden.

`Row.isSolution()` funktioniert folgendermaßen: alle möglichen Versätze werden durchlaufen. Sollte dabei irgendwo das Abbruchkriterium erfüllt sein (wenn Versatz 15, 16 oder die Anzahl der Teilnehmer ist), wird `true` zurückgegeben. Wenn dies nicht der Fall ist, wird `false` zurückgegeben.

Folgendes geschieht in der Methode `Row.printSolution()`: zunächst wird eine Variable `syllables` erstellt, in welcher gespeichert wird, wie oft das Geburtstagskind nur eine Silbe sagen soll. Außerdem wird der neuen Variablen `solution` eine neue Instanz von `ArrayList<Boolean>` zugewiesen. Dies soll die Lösung sein und beinhaltet für jedes Mal, wenn das Geburtstagskind nur eine Silbe sagt, `false`, wenn sie jedoch zwei Silben sagt, kommt in diese Liste `true` rein. Nun werden alle möglichen Versätze mit einer `for`-Schleife durchlaufen und für den ersten Versatz, der gefunden wird, wird die Lösung erstellt. Das bedeutet, dass bei einem Fund die Schleife mit einem `break`-Statement abgebrochen wird. Davor wird die Variable syllables auf den entsprechenden, mit der zugehörigen Formel berechneten, Wert gesetzt. Sollte 15 oder 16 gefunden worden sein, muss, wie unter „Lösungsidee“ beschrieben, noch `true` bzw. `false` in die Lösung `solution` eingefügt werden.
Durch eine weitere `for`-Schleife wird nun `syllables` Mal `false` in die Liste getan. Außerdem wird so oft `true` in die Liste getan, wie die Formel unter „Lösungsidee“ aussagt.
Abschließend wird das Ergebnis in der Konsole angezeigt.

Beispiele

Beispiel 1

```
Bitte die Anzahl von Teilnehmern eingeben:
28
[true, false, false, false, false, true]
```

Dies ist das Beispiel, nach welchem in der Aufgabe gefragt ist. Dabei nehmen 28 Teilnehmer an der Buffet-Lotterie teil. Um möglichst schnell essen gehen zu können, sollte das Geburtstagskind vier Mal eine Silbe und zwei Mal zwei Silben sagen. Dabei ist, wie unter dem Abschnitt „Lösungsidee“ bereits beschrieben, die Reihenfolge, *wann* das Geburtstagskind wie viele Silben sagt, egal.

Beispiel 2

```
Bitte die Anzahl von Teilnehmern eingeben:
18
[false, false, false, false, false, true, true]
```

Zu dieser Anzahl der Teilnehmer wurde bereits am Anfang dieser Dokumentation die Tabelle aufgestellt. 18 Teilnehmer gibt es, das Geburtstagskind sollte fünf Mal eine Silbe und zwei Mal zwei Silben sagen.

Beispiel 3

```
Bitte die Anzahl von Teilnehmern eingeben:
2
[false, false, false, false, false, false, false, true]
```

An dieser Buffet-Lotterie nehmen lediglich zwei Teilnehmer teil. Um als Erster essen gehen zu können, sollte das Geburtstagskind sieben Mal eine Silbe und ein Mal zwei Silben sagen. Es wäre auch möglich, dass das Geburtstagskind mehrmals zwei Silben und seltener nur eine Silbe sagt, wodurch sich die beiden Teilnehmer nicht ganz so oft abwechseln würden. Jedoch werden dabei auch genau 16 Silben gesagt, deshalb gehen beide Varianten gleich schnell.

Beispiel 4

```
Bitte die Anzahl von Teilnehmern eingeben:
1000000
[true, true, true]
```

Dieses Beispiel zeigt, dass auch die Berechnung mit sehr großen Zahlen, in diesem Fall 1000000, einfach funktioniert. Dabei sollte das Geburtstagskind 3 Mal 2 Silben sagen.

Quelltext

`Row` — Stellt eine Zeile dar.

```java
public class Row {

    private int alteAnzahl;
    private int versatz;
    private int versatzLaenge;
    private int raus;

    // ...

}
```

`Row` — Konstruktor und Getter-Methoden.

```
public Row(int alteAnzahl, int versatz, int versatzLaenge, int raus) {
    this.alteAnzahl = alteAnzahl;
    this.versatz = versatz;
    this.versatzLaenge = versatzLaenge;
    this.raus = raus;
}

public int getAlteAnzahl() {
    return alteAnzahl;
}

public int getVersatz() {
    return versatz;
}

public int getVersatzLaenge() {
    return versatzLaenge;
}

public int getRaus() {
    return raus;
}
```

`Row.printSolution()`

```
/**
 * Gibt diese Lösung in die Konsole aus.
 */
public void printSolution() {
    int syllables = 0; // die Anzahl, wie oft false in der Lösung auftauchen soll

    List<Boolean> solution = new ArrayList<Boolean>(); // die Lösung als
    // List, welche zum Schluss ausgegeben wird

    // Versatz wird durchlaufen. Wenn eine Lösung 15 oder 16 gefunden wurde,
    // muss noch true bzw. false angehangen werden, damit auch das
    // Geburtstagskind die letzte Silbe sagt. Wenn die Lösung der alten
    // Anzahl der Teilnehmer entspricht, darf kein Ende angehangen werden,
    // da die letzte Silbe bereits vom Geburtstagskind gesagt wird.
    for (int i = getVersatz(); i < getVersatz() + getVersatzLaenge(); i++) {
        if (i == getAlteAnzahl()) {
            syllables = getVersatz() + getVersatzLaenge() - getAlteAnzahl() - 1;
            break;
        } else if (i == 15) {
            solution.add(true);
            syllables = getVersatz() + getVersatzLaenge() - 16;
            break;
        } else if (i == 16) {
            solution.add(false);
            syllables = getVersatz() + getVersatzLaenge() - 17;
            break;
        }
    }

    // Silben werden der Liste angefügt. syllables gibt an, wie oft false
    // verwendet wird.
    for (int i = 1; i < getVersatzLaenge(); i++)
        if (syllables == 0)
            solution.add(true);
        else {
            syllables--;
            solution.add(false);
        }

    System.out.println(solution);
}
```

Row.isSolution()

```java
/**
 * Prüft, ob die Zeile eine Lösung darstellt. Dies ist nur der Fall, wenn
 * eine Zahl vom Versatz bis zum Versatz insklusive der Länge des Versatzes
 * 15, 16 oder der alten Anzahl entspricht.
 *
 * @return {@code true}, wenn diese Zeile eine Lösung ist, {@code false},
 * wenn nicht
 */
public boolean isSolution() {
    for (int i = getVersatz(); i < getVersatz() + getVersatzLaenge(); i++)
        if (i == 15 || i == 16 || i == getAlteAnzahl())
            return true;
    return false;
}
```

BuffetLotterieGenerator

```java
public class BuffetLotterieGenerator {

    /**
     * Die Anzahl der Silben im Abzählvers.
     */
    private final static int SYLLABLES = 16;

    /**
     * Die Anzahl der Teilnehmer beim Start.
     * Diese Variable wird nie geändert.
     */
    private int participants;

    /**
     * Erstellt eine neue Instanz dieser Klasse
     * @param participants die Anzahl der Teilnehmer beim Start
     */
    public BuffetLotterieGenerator(int participants) {
        this.participants = participants;
    }

    // ...

}
```

BuffetLotterieGenerator.e()

```java
/**
 * Berechnet die Anzahl der Teilnehmer, die essen gehen.
 *
 * @param row die letzte Zeile
 * @param a   alte Anzahl der Teilnehmer
 * @return die Anzahl der Teilnehmer, die essen gehen
 */
private int e(Row row, int a) {
    // Am Anfang sind noch alle Teilnehmer drin:
    if (row == null)
        return participants / SYLLABLES;

    // Ansonsten wird die neue Anzahl der Teilnehmer berechnet, wobei der
    // Versatz hnzugefügt wird, das Ergebnis wird durch die Silben geteilt:
    return (row.getVersatz() + a) / SYLLABLES;
}
```

BuffetLotterieGenerator.generate()

```
/**
 * Generiert die Lösung.
 * @return die letzte Zeile der Lösung
 */
public Row generate() {
    Row row = null;

    // Wird fortgesetzt beim ersten Versuch (row == null) und wenn die letzte
    // Zeile noch keine Lösung enthält.
    while (row == null || !row.isSolution()) {

        int a = row == null ? participants : (row.getAlteAnzahl() - row.getRaus());
        // Alte Anzahl der Teilnehmer. In der ersten Zeile ist dies die
        // Gesamtzahl, ansonsten wird die letzte Anzahl genommen und davon
        // subtrahiert, wie viele Teilnehmer essen gehen.

        int e = e(row, a); // e: Anzahl der Teilnehmer, die rausgehen
        if (e == 0) // mindestens ein Teilnehmer muss rausgehen
            e = 1;

        int versatz = versatz(row, a, e); // kleinster Versatz
        int letzterVersatz = letzterVersatz(row, a, e); // größter Versatz

        int versatzLaenge = row == null ? 2 : row.getVersatzLaenge() + 1;
        // versatzLaenge ist beim ersten Durchgang zwei, danach steigt diese
        // immer um 1

        // Geburtstagskind wurde ein weiteres Mal "überrundet":
        while (versatz <= 0) {

            // Wenn auch der letzte Versatz das Geburtstagskind überrundet wurde:
            if (letzterVersatz <= 0) {
                // Geburtstagskind kann sich 1 Mal mehr entscheiden:
                versatzLaenge++;
                // weitere Runde gestartet:
                letzterVersatz += a;
            }

            versatz += a;
            // zum Versatz wird die Anzahl der Teilnehmer hinzugefügt, da eine
            // weitere Runde gestartet wird
        }

        row = new Row(a, versatz, versatzLaenge, e);
    }

    return row; // Zeile enthält Lösung, kann zurückgegeben werden
}
```

`BuffetLotterieGenerator.versatz()`
`BuffetLotterieGenerator.letzterVersatz()`

```java
/**
 * Berechnet den neuen Versatz.
 * @param row die letzte Zeile
 * @param a alte Anzahl der Teilnehmer
 * @param e Anzahl der Teilnehmer, die essen gehen
 * @return den neuen Versatz
 */
private int versatz(Row row, int a, int e) {
    return a - ((SYLLABLES * e) - (row == null ? 1 : row.getVersatz()));
    // Formel: v' = a - (16e - v)
    // Wenn die erste Zeile berechnet wird, ist v = 1
}

/**
 * Berechnet den letzten Versatz.
 * @param row die letzte Zeile
 * @param a alte Anzahl der Teilnehmer
 * @param e Anzahl der Teilnehmer, die essen gehen
 * @return den letzten Versatz
 */
private int letzterVersatz(Row row, int a, int e) {
    return a - ((SYLLABLES * e) - (row == null ? 1 :
            row.getVersatz() + row.getVersatzLaenge() - 1));
    // Formel: v' = a - (16e - v)
    // v ist dabei der letzte Versatz, nicht der kleinste
}
```

Mit dem folgenden Code wird das Programm in der Klasse `BuffetLotterie` gestartet, nachdem die Teilnehmerzahl `participants` eingelesen wurde.

```java
new BuffetLotterieGenerator(participants).generate().printSolution();
```